БЕН ВАТСОН

C# 4.0

НА ПРИМЕРАХ

НОВЫЕ ВОЗМОЖНОСТИ C# 4.0 И .NET 4.0.

ОТ ОСНОВ ЯЗЫКА ДО РЕШЕНИЯ ТИПИЧНЫХ ЗАДАЧ

БОЛЕЕ 200 ПРОВЕРЕННЫХ РЕШЕНИЙ, ОПТИМАЛЬНЫХ ТЕХНИЧЕСКИХ ПРИЕМОВ И ПРИМЕРОВ ПРОГРАММНОГО КОДА

SAMS

bhv®

BEN WATSON

C# 4.0

HOW-TO

SAMS | 800 East 96th Street, Indianapolis, Indiana 46240 USA

Бен Ватсон

C# 4.0
на примерах

Санкт-Петербург
«БХВ-Петербург»
2011

УДК 681.3.068+800.92C#
ББК 32.973.26-018.1
В21

Ватсон Б.

В21 C# 4.0 на примерах. — СПб.: БХВ-Петербург, 2011. — 608 с.: ил.

ISBN 978-5-9775-0608-3

На практических примерах рассмотрено программирование на языке Microsoft C# 4.0, начиная с основ языка и заканчивая решением типичных задач с помощью .NET Framework. Показано создание эффективных классов, интерфейсов и типов, а также программного кода, допускающего многократное использование. Описаны приемы обработки данных, основанные на применении коллекций, сериализации, баз данных и XML. Рассмотрена реализация пользовательского интерфейса с применением технологий WinForms и WPF, а также создание веб-приложений на основе технологий ASP.NET и Silverlight. Показано применение на практике новых возможностей языка C# 4.0. Уделено внимание взаимодействию с ОС Windows и системным ПО, использованию шаблонов для разработки сложных программ и др. Приведено более 200 готовых решений, оптимальных технических приемов и примеров проверенного кода.

Для программистов

УДК 681.3.068+800.92C#
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Перевод с английского	*Сергея Иноземцева*
Редактор	*Ирина Иноземцева*
Компьютерная верстка	*Натальи Караваевой*
Корректор	*Виктория Пиотровская*
Дизайн серии	*Игоря Цырульникова*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 30.09.10.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 49,02.
Тираж 1700 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.60.953.Д.005770.05.09 от 26.05.2009 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12.

ISBN ISBN 978-0-672-33063-6 (англ.)
ISBN 978-5-9775-0608-3 (рус.)

Оглавление

Моим родителям, Майку и Диане Ватсон, которые с пониманием отнеслись к моей ребяческой просьбе купить компилятор C++ и всемерно поощряли мои интересы и развивали мои способности. Я ничего бы не добился без их личного примера, любви и советов.

Моей замечательной жене Летиции, проявившей громадное терпение и большую любовь, когда я взялся за этот новый и пугающий проект на фоне перемен, произошедших в нашей жизни.

Об авторе

Бен Ватсон (Ben Watson) программирует на платформе .NET Framework с первых дней ее существования. До перехода в корпорацию Microsoft он работал ведущим программистом в GeoEye, компании по обработке спутниковых изображений, где создавал системы морской связи на платформе .NET. Последнее время он трудился в команде разработчиков Query Pipeline для Bing, участвуя в реализации масштабируемых распределенных систем и других компонентов этой поисковой службы.

Благодарности

Я очень благодарен тем, кто приложил все свои усилия, помогая мне работать над этой книгой.

Бруку Фарлингу (Brook Farling), редактору, который нашел меня и предоставил мне возможность написать книгу, я благодарен за то, что он провел большую координаторскую работу, проявил огромное терпение и вложил много энергии в доведение дела до конца.

Марку Стромайеру (Mark Strawmyer), техническому редактору, я говорю спасибо за то, что он прочитал внушительный объем текста и кода с целью обеспечения высокого качества материала.

Благодарю всю команду издательства Sams, участвовавшую в подготовке книги к публикации. Это Марк Ренфроу (Mark Renfrow), Лори Лайонз (Lori Lyons), Барт Рид (Bart Reed), Нони Рэтклиф (Nonie Ratcliff) и Шери Кейн (Sheri Cain).

Введение

Краткий обзор книги

Подход, предпринятый в этой книге, весьма далек от "библейского", охватывающего максимум материала по теме. Книга имеет структуру "сборника рецептов", то есть состоит из типичных задач и легко воспроизводимых способов их решения. Как автор, я стремился свести к минимуму теоретические пояснения и сосредоточиться преимущественно на самом коде. Многие неочевидные моменты разъясняются прямо в комментариях к коду.

Строго говоря, книга не является описанием языка. Наряду с обсуждением функциональных возможностей языка материал включает в себя конкретные примеры приложений, полезные алгоритмы и советы, применимые во многих ситуациях.

Программисты, как начинающие, так и опытные, найдут в книге сотни интересных тем, таких как малоизвестные операторы C#, сортировка строк, содержащих числа, или реализация функции "Отмена" в пользовательском интерфейсе. И каждая тема сопровождается программными решениями, которые пригодятся в самых разных ситуациях, независимо от профессионального уровня разработчика.

Короче говоря, я написал книгу, которую сам хотел бы иметь под рукой в то время, когда еще изучал основы программирования на языке C#, и сейчас, когда мне требуется получить справку или освежить в памяти тот или иной технический прием.

Как извлечь максимум пользы из этой книги

Мы создали книгу, которую легко читать "от корки до корки". Наша цель — дать читателю полное представление о языке C# 4.0. Материал книги поделен на четыре части, содержащие главы, которые легко читать и среди которых легко найти нужную.

Часть I, "Основы программирования на языке C#", посвящена наиболее общим функциональным возможностям языка C#, которые важны при решении любых программистских задач. Хотя приведенные сведения могут показаться элементарными, в главах этой части содержится множество полезных советов, помогающих досконально овладеть темой.

- *Глава 1. "Основы работы с типами"*
- *Глава 2. "Создание типов с разносторонней функциональностью"*

- *Глава 3. "Общие принципы кодирования"*
- *Глава 4. "Исключения"*
- *Глава 5. "Числа"*
- *Глава 6. "Перечисления"*
- *Глава 7. "Строки"*
- *Глава 8. "Регулярные выражения"*
- *Глава 9. "Универсальные типы"*

Часть II, "Обработка данных", содержит сведения о том, как хранить и обрабатывать данные, в том числе и полученные из Интернета.

- *Глава 10. "Коллекции"*
- *Глава 11. "Файлы и сериализация"*
- *Глава 12. "Работа в сетях и во Всемирной паутине"*
- *Глава 13. "Базы данных"*
- *Глава 14. "Язык XML"*

Часть III, "Взаимодействие с пользователем", затрагивает наиболее популярные парадигмы пользовательского интерфейса на платформе .NET, причем речь идет как о локально работающих программах, так и о веб-приложениях.

- *Глава 15. "Делегаты, события и анонимные методы"*
- *Глава 16. "Технология Windows Forms"*
- *Глава 17. "Графика с применением Windows Forms и GDI+"*
- *Глава 18. "WPF"*
- *Глава 19. "ASP.NET"*
- *Глава 20. "Silverlight"*

Часть IV, "Более сложные элементы языка C#", содержит материал о нетривиальных возможностях языка C#, позволяющих перевести ваши приложения на более высокий уровень в том, что касается их производительности, шаблонов проектирования, полезных алгоритмов и т. д.

- *Глава 21. "LINQ"*
- *Глава 22. "Управление памятью"*
- *Глава 23. "Потоки выполнения. Асинхронное и параллельное программирование"*
- *Глава 24. "Отражение и создание добавляемых модулей"*
- *Глава 25. "Шаблоны приложений и полезные советы по проектированию"*
- *Глава 26. "Взаимодействие с операционной системой и аппаратной частью"*
- *Глава 27. "Полезные мелочи"*
- *Приложение. "Необходимые инструменты"*

Весь приведенный код был создан в предварительных выпусках Visual Studio 2010, но в большинстве случаев вы можете использовать более ранние версии, особенно для написания кода, не связанного с .NET 4. Если у вас нет Visual Studio, вы можете загрузить издание Express Edition, доступное по адресу **www.microsoft.com/express/default.aspx**. Эта версия позволит построить почти любой пример, код которого приведен в книге.

Вы можете зарегистрироваться на сайте книги. Зайдите на страницу **informit.com/register**, выполните процедуру бесплатной регистрации и укажите ISBN-номер книги[1]. Затем загляните на страницу **Account** (Учетная запись) и в разделе **Registered Products** (Зарегистрированные товары) найдите ссылку **Access Bonus Content** (Получить доступ к бонусному содержимому). Вы получите доступ к обновлениям и загрузочным файлам, если таковые имеются для этой книги, а также к списку замеченных ошибок.

Как углубить и расширить свои знания

Невозможно написать книгу, охватывающую все, что связано с C# и .NET Framework. Пожалуй, даже нельзя надеяться на полное раскрытие хотя бы одной темы из этой области. Если такая книга все-таки была бы написана, то никто не смог бы поднять ее и, тем более, прочитать за разумное время.

Когда вы овладеете основами, в вашем распоряжении окажется множество ресурсов, где вы найдете ответы на возникающие вопросы и получите более глубокие знания о платформе .NET.

К счастью, MSDN-документация по платформе .NET написана очень хорошо (она доступна по адресу **http://msdn.microsoft.com/en-us/library/aa139615.aspx**[2]). Большинство ее разделов включает в себя код и разъяснения по его использованию. В качестве дополнительного бонуса в конце многих разделов можно найти полезные советы, оставленные другими разработчиками.

Форумы разработчиков .NET (**http://social.msdn.microsoft.com/Forums/en-US/category/netdevelopment**[3]) — это отличное место для того, чтобы задать вопрос. Вам ответят эксперты, многие из которых на практике занимались разработкой и тестированием .NET.

Кроме того, я нашел в Интернете отличное место, где можно получить ответы на интересующие вас вопросы: **StackOverflow.com**.

На мой взгляд, самым лучшим советом по поводу расширения и углубления знаний будет такой: пишите программы. Занимайтесь этим постоянно, обдумывайте новые проекты, применяйте новые технологии, стремитесь выйти за пределы своих возможностей. Вообще говоря, книги полезны лишь до определенного момента, после чего вы должны будете приступить к написанию кода. И тогда эта книга станет надежным справочным руководством, незаменимым в тех случаях, когда вы не знаете, как подступиться к решению задачи.

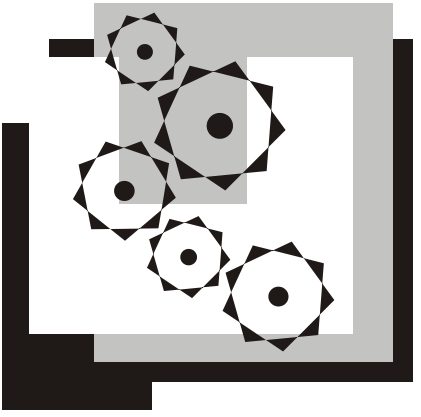
Желаю вам успехов в программировании.

---

[1] Сайт **informit.com** англоязычный. Следует указывать ISBN-номер оригинального издания: 978-0-672-33063-6 — *прим. ред.*

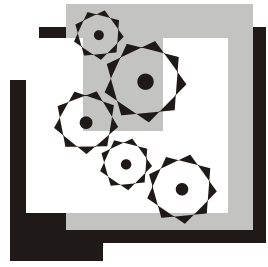
[2] Документация на русском языке находится по адресу **http://msdn.microsoft.com/ru-ru/library/aa139615.aspx** — *прим. перев.*

[3] Русскоязычный форум: **http://social.msdn.microsoft.com/Forums/ru-RU/category/netdevelopment** — *прим. перев.*



ЧАСТЬ I

Основы программирования на языке C#



Глава 1

Основы работы с типами

В этой главе разъясняются основы создания типов в языке C#. Если вы уже знакомы с ним, большая часть этой главы не содержит ничего нового для вас.

После обсуждения членов класса, таких как поля, свойства и методы, вы прочитаете о конструкторах, о том, как создавать и реализовывать интерфейсы, и о том, когда следует использовать структуры.

Информация в этой главе элементарна, но очень важна. Как и во многих ситуациях, если вы не овладеете основами, у вас ничего не получится.

Создание класса

Задача. Вам необходимо написать объявление класса.

Решение. Начнем с объявления простого класса, содержащего координаты точки в трехмерном пространстве:

```
// Необходимо импортировать пространства имен по умолчанию,
// которые Visual Studio включит в каждый файл
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ClassSample
{
  // Использован модификатор public,
  // чтобы класс был виден за пределами сборки
  public class Vertex3d
  {
  }
}
```

Класс, который мы объявили, пока пуст, но мы заполним его по ходу этой главы.

Класс определен как открытый, то есть он виден каждому типу, ссылающемуся на сборку. В языке C# существует несколько модификаторов доступа, и они собраны в табл. 1.1.

Таблица 1.1. *Модификаторы доступа*

Модификатор	Применяется к	Описание
`Public`	типам, членам	доступен любому коду, даже за пределами сборки
`Private`	типам, членам	доступен коду в том же типе
`Internal`	типам	доступен коду в той же сборке
`Protected`	членам	доступен коду в том же или в производном типе
`Protected internal`	членам	доступен коду в той же сборке или в производном классе в другой сборке

Если видимость класса не указана, по умолчанию предполагается модификатор `internal`.

Определение полей, свойств и методов

Задача. Вам нужно добавить поля, свойства и методы в определение класса.
Решение. Придадим некоторую осмысленность классу `Vertex3d`:

```
public class Vertex3d
{
  // Поля
  private double _x;
  private double _y;
  private double _z;
  // Свойства
  public double X
  {
    get { return _x; }
    set { _x = value; }
  }
  public double Y
  {
    get { return _y; }
    set { _y = value; }
  }
  public double Z
  {
    get { return _z; }
    set { _z = value; }
  }
  // Метод
```

```
  public void SetToOrigin()
  {
    X = Y = Z = 0.0;
  }
}
```

Сделаем несколько замечаний относительно этого кода.

- Поля объявлены с модификатором `private`, что, вообще говоря, является хорошим стилем.
- Свойства объявлены как `public`, но могли быть `private`, `protected` или `protected internal`, по желанию программиста.
- Свойства могут включать в себя метод `get` или `set` (или оба).
- В свойствах (то есть в коде) подразумевается наличие аргумента `value`.

В следующем примере значение `13.0` будет передано свойству `X` через аргумент `value`:

```
Vertex3d v = new Vertex3d();
v.X = 13.0;
```

Применение автоматически реализуемых свойств

Вы часто будете встречать код такого вида:

```
class MyClass
{
  private int _field = 0;
  public int Field { get { return _field; } set { _field = value; } }
}
```

В языке C# для этой конструкции имеется сокращенный вариант:

```
class MyClass
{
  public int Field {get; set;}
  // Теперь необходимо инициализировать value в конструкторе
  public MyClass()
  {
    this.Field = 0;
  }
}
```

Примечание

Нельзя иметь автоматически реализуемое свойство при наличии только метода `get` (в самом деле, если нет поля с резервным значением, как вы установите свойство?). Однако можно иметь закрытый (`private`) метод `set`:

```
public int Field { get ; private set; }
```

Определение статических членов

Задача. Вам нужно определить данные или методы, применимые к типу, а не отдельным экземплярам. Они часто используются в методах, действующих на несколько экземпляров типа.

Решение. Воспользуйтесь модификатором `static`, как сделано в следующем методе для создания двух объектов `Vertex3d`:

```
public class Vertex3d
{
  ...
  public static Vertex3d Add(Vertex3d a, Vertex3d b)
  {
    Vertex3d result = new Vertex3d();
    result.X = a.X + b.X;
    result.Y = a.Y + b.Y;
    result.Z = a.Z + b.Z;
    return result;
  }
}
```

Этот статический метод вызывается так:

```
Vertex3d a = new Vertex3d(0,0,1);
Vertex3d b = new Vertex3d(1,0,1);
Vertex3d sum = Vertex3d.Add(a, b);
```

Написание конструктора

Задача. Вам необходима автоматическая инициализация создаваемых объектов класса.

Решение. Определите специальный метод, называемый *конструктором*. Он должен носить то же имя, что и класс, но не возвращать никакого значения. Конструктор работает во время создания типа, — он никогда не вызывается напрямую.

Рассмотрим два конструктора для класса `Vertex3d`: один имеет аргументы, а другой выполняет инициализацию значением по умолчанию.

```
class Vertex3d
{
  public Vertex3d()
  {
    _x = _y = _z = 0.0;
  }
  public Vertex3d(double x, double y, double z)
  {
    this._x =x;
```

```
    this._y = y;
    this._z = z;
  }
}
```

Примечание

Конструкторам необязательно быть открытыми. Например, можно иметь защищенный (`protected`) конструктор, доступный только из производных классов. Более того, вы можете создать закрытый конструктор, предотвращающий создание экземпляров (для утилитных классов), или доступный только из других методов того же класса (возможно, статических генерирующих методов).

Создание статического конструктора и выполнение инициализации

Задача. У класса есть статические данные, подлежащие инициализации.

Решение. Статические поля можно инициализировать двумя способами. Первый заключается в использовании статического конструктора, аналогичного стандартному, но лишенного модификатора доступа и аргументов:

```
public class Vertex3d
{
  private static int _numInstances;
  static Vertex3d()
  {
    _numInstances = 0;
  }
  ...
}
```

Однако для повышения производительности предпочтительнее инициализировать статические поля в коде, когда это возможно:

```
public class Vertex3d
{
  private static int _numInstances = 0;
  ...
}
```

Инициализация свойств при конструировании

Задача. Вы хотите инициализировать свойства класса в момент создания переменной, даже если у конструктора нет для этого аргументов.

Решение. Воспользуйтесь синтаксисом инициализации объекта, как показано в следующем примере:

```
class Person
{
```

```
  public int Id { get; set; }
  public string Name { get; set; }
  public string Address { get; set; }
}
...
Person p = new Person()
              { Id = 1, Name = "Ben", Address = "Redmond, WA" };
```

Применение модификаторов *const* и *readonly*

Задача. Требуется определить поля, значения которых не могут быть изменены на этапе выполнения.

Решение. Для определения неизменяющихся данных служат модификаторы `const` и `readonly`. Между ними есть важное различие: поля с модификатором `const` должны быть определены при объявлении. По этой причине и в силу факта неизменности их значений подразумевается, что они принадлежат типу как статические поля. Что касается полей с модификатором `readonly`, они могут быть заданы при объявлении или в конструкторе, но больше нигде.

```
public class Vertex3d
{
  private const string Name = "Vertex";
  private readonly int ver;
  public Vertex3d()
  {
    ver = Config.MyVersionNumber;// Ok
  }
  public void SomeFunction()
  {
    ver = 13;// Ошибка!
  }
}
```

Повторное использование кода в нескольких конструкторах

Задача. У вас имеется несколько конструкторов, обладающих сходной функциональностью в некоторых частях. Вы хотите избежать дублирования кода. В языке C++ и в некоторых старых языках нередко возникала ситуация, в которой класс с несколькими конструкторами должен был вызывать один и тот же код инициализации. Тогда общий код обычно оформлялся в виде функции, которую вызывал каждый конструктор.

```
// Пример на языке C++
class MyCppClass
```

```
{
public:
  MyCppClass() { Init(); }
  MyCppClass(int arg) { Init(); }
private:
  void Init() { /* Здесь код инициализации*/ };
}
```

Решение. В языке C# можно вызывать другие конструкторы в пределах класса, для чего используется ключевое слово `this`:

```
public class Vertex3d
{
  public Vertex3d(double x, double y, double z)
  {
    this._x = x;
    this._y = y;
    this._z = z;
  }
  public Vertex3d(System.Drawing.Point point)
    :this(point.X, point.Y, 0)
  {
  }
  ...
}
```

Создание производного класса

Задача. Вам нужно специализировать класс, добавив новые или переопределив имеющиеся функциональные возможности.

Решение. Добавьте новую функциональность к коду базового класса с помощью наследования.

```
public class BaseClass
{
  private int _a;
  protected int _b;
  public int _c;
}
public class DerivedClass : BaseClass
{
  public DerivedClass()
  {
   _a = 1;// Так нельзя! Переменная имеет модификатор private в BaseClass
   _b = 2;// ok
```

```
    _c = 3;// ok
  }
  public void DoSomething()
  {
    _c = _b = 99;
  }
}
```

При создании производного класса сохраняется доступ к открытым и защищенным членам базового класса, но не к его закрытым членам.

Вызов конструктора базового класса

Задача. У вас есть класс-потомок, и вы хотите, чтобы его конструктор вызывал конкретный конструктор базового класса.

Решение. Аналогично тому, как из конструктора класса вызываются другие конструкторы, можно вызывать и конструкторы базового класса. Если конструктор не уточняется, будет вызван конструктор базового класса, установленного по умолчанию. Если он имеет аргументы, их нужно будет указать.

```
public class BaseClass
{
  public BaseClass(int x, int y, int z)
  { ... }
}
public class DerivedClass : BaseClass
{
  public DerivedClass()
  : base(1, 2, 3)
  {
  }
}
```

Переопределение метода или свойства базового класса

Задача. Вам нужно переопределить поведение базового класса в его потомке.

Решение. Нужно объявить метод или свойство базового класса с модификатором `virtual`, и этот член должен быть доступен из производного класса. У производного класса должен быть модификатор `override`.

```
public class Base
{
  Int32 _x;
  public virtual Int32 MyProperty
```

```
  {
    get
    {
      return _x;
    }
  }
  public virtual void DoSomething()
  {
    _x = 13;
  }
}
public class Derived : Base
{
  public override Int32 MyProperty
  {
    get
    {
      return _x * 2;
    }
  }
  public override void DoSomething()
  {
    _x = 14;
  }
}
```

Из базового класса можно ссылаться на его экземпляры и на экземпляры любого прозводного класса. Например, следующий код выведет 28, а не 13:

```
Base d = new Derived();
d.DoSomething();
Console.WriteLine(d.MyProperty().ToString());
```

Кроме того, вы можете вызывать функции базового класса из прозводного класса. Для этого пользуйтесь ключевым словом `base`.

```
public class Base
{
  public virtual void DoSomething()
  {
    Console.WriteLine("Base.DoSomething”);
  }
}
public class Derived
{
  public virtual void DoSomething()
```