

原版风暴 · 开发大师系列



C# Primer A Practical Approach

畅销书作者 Stan Lippman
继《C++ Primer》后最新作品

C# Primer

(影印版)

[美] Stanley B. Lippman 著



中国电力出版社

www.infopower.com.cn

原履风暴·开发大师系列

C# Primer A Practical Approach

C# Primer

(影印版)

[美] Stanley B. Lippman 著

中国电力出版社

C# Primer A Practical Approach (ISBN 0-201-72955-5)

Stanley B. Lippman

Copyright © 2002 Addison Wesley Longman, Inc.

Original English Language Edition Published by Addison Wesley Longman, Inc.

All rights reserved.

Reprinting edition published by PEARSON EDUCATION ASIA LTD and CHINA ELECTRIC POWER PRESS, Copyright © 2003.

本书影印版由 Pearson Education 授权中国电力出版社在中国境内（香港、澳门特别行政区和台湾地区除外）独家出版、发行。

未经出版者书面许可，不得以任何方式复制或抄袭本书的任何部分。

本书封面贴有 Pearson Education 防伪标签，无标签者不得销售。

北京市版权局著作合同登记号：图字：01-2003-3824

图书在版编目（CIP）数据

C# Primer / (美) 利普曼著. —影印本. —北京：中国电力出版社，2003

（原版风暴·开发大师系列）

ISBN 7-5083-1090-X

I. C... II. 利... III. C 语言—程序设计—英文 IV. TP312

中国版本图书馆 CIP 数据核字（2003）第 067671 号

出版者：中国电力出版社（北京西城区三里河路 6 号，邮编 100044） <http://www.infopower.com.cn>

责任编辑：陈维宁

丛书名：原版风暴·开发大师系列

书名：C# Primer（影印版）

编著：（美）Stanley B. Lippman

出版者：中国电力出版社

地址：北京市三里河路6号 邮政编码：100044

电话：（010）88515918 传 真：（010）88518169

印 刷：汇鑫印务有限公司

发 行 者：新华书店总店北京发行所

开 本：787×1092 1/16 印 张：26

书 号：ISBN 7-5083-1090-X

版 次：2003年9月北京第一版

印 次：2003年9月第一次印刷

定 价：48.00 元

Beth

*Imagine, we have shared a lifetime together.
Thanks for understanding
and being there*

Danny

*Hey, dude. Wassup?
So this is what I've been doing—
I know you thought I just didn't want to help with your algebra ...*

Anna

*Whoa. It's really done. I know, finally.
A slew of IOU's:
Legoland, the batting cage, Hogwarts ...*

*And in loving memory of
George and Ray Lippman*

Preface

C# is a new language invented at Microsoft and introduced with Visual Studio.NET. More than a million lines of C# code already have gone into the implementation of the .NET class framework. This book covers the C# language and its use in programming the .NET class framework, illustrating application domains such as ASP.NET and XML.

My general strategy in presenting the material is to introduce a programming task and then walk through one or two implementations, introducing language features or aspects of the class framework as they prove useful. The goal is to demonstrate how to use the language and class framework to solve problems rather than simply to list language features and the class framework API.

Learning C# is a two-step process: learning the details of the C# language and then becoming familiar with the .NET class framework. This two-step process is reflected in the organization of this text.

In the first step we walk through the language—both its mechanisms, such as class and interface inheritance and delegates, and its underlying concepts, such as its unified type system, reference versus value types, *boxing*, and so on. This step is covered in the first four chapters.

The second step is to become familiar with the .NET class framework, in particular with Windows and Web programming and the support for XML. This is the focus of the second half of the book.

Working your way through the text should jump-start your C# programming skills. In addition, you'll become familiar with a good swatch of the .NET class framework. All the program code is available for download at my company's Web site www.objectwrite.com.

Mail can be sent to me directly at slippman@objectwrite.com.

Organization of the Book

The book is organized into eight relatively long chapters. The first four chapters focus on the C# language, looking at the built-in language features, the class mechanism, class inheritance, and interface inheritance. The second four chapters explore the various library domains supported within the .NET class framework.

Chapter 1 covers the basic language, as well as some of the fundamental classes provided within the class framework. The discussion is driven by the design of a small program. Concepts such as namespaces, exception handling, and the unified type system are introduced.

Chapter 2 covers the fundamentals of building classes. We look at access permission, distinguish between `const` and `readonly` members, and cover specialized methods such as indexers and properties. We walk through the different strategies of member initialization, as well as the rules for operator overloading and conversion operators. We look at the `delegate` type, which serves as a kind of universal pointer to a function.

Chapters 3 and 4 cover, in turn, class and interface inheritance. Class inheritance allows us to define a family of specialized types that override a generic interface, such as an abstract `WebRequest` base class and a protocol-specific `HttpWebRequest` subtype. Interface inheritance, on the other hand, allows us to provide a common service or shared attribute for otherwise unrelated types. For example, the `IDisposable` interface frees resources. Classes holding database connections or window handles are both likely to implement `IDisposable`, although they are otherwise unrelated.

Chapter 5 provides a wide-ranging tour of the .NET class library. We look at input and output, including file and directory manipulation, regular expressions, sockets and thread programming, the `WebRequest` and `WebResponse` class hierarchies, a brief introduction to ADO.NET and establishing database connections, and the use of XML.

Chapters 6 and 7 cover, in turn, drag-and-drop Windows Forms and Web Forms development. Chapter 7 focuses on ASP.NET, and the Web page life cycle. Both chapters provide lots of examples of using the prebuilt controls and attaching event handlers for user interaction.

The final chapter provides a programmer's introduction to the .NET Common Language Runtime. It focuses on assemblies, type reflection, and attributes, and concludes with a brief look at the underlying intermediate language that is the compilation target of all .NET languages.

Written for Programmers

The book does not assume that you know C++, Visual Basic, or Java. But it does assume that you have programmed in some language. So, for example, I don't assume that you know the exact syntax of the C# `foreach` loop statement, but I do assume that you know what a loop is. Although I will illustrate how to invoke a function in C#, I assume you know what I mean when I say we "invoke a function." This text does not require previous knowledge of object-oriented programming or of the earlier versions of ASP and ADO.

Some people—some very bright people—argue that under .NET, the programming language is secondary to the underlying Common Language Runtime (CLR) upon which the languages float like the continents on tectonic plates. I don't agree. Language is how we express ourselves, and the choice of one's language affects the design of our programs. The underlying assumption of this book is that C# is the preferred language for .NET programming.

The book is organized into eight relatively long chapters. The first set of four chapters focuses on the C# language, looking at the built-in language features, the class mechanism, class inheritance, and interface inheritance. The second set of four chapters explores the various library domains supported within the .NET class framework, such as regular expressions, threading, sockets, Windows Forms, ASP.NET, and the Common Language Runtime.

Lexical Conventions

Type names, objects, and keywords are set off in Courier font, as in `int`, a predefined language type; `Console`, a class defined in the framework; `maxCount`, an object defined either as a data member or as a local object within a function; and `foreach`, one of the predefined loop statements. Function names are followed by an empty pair of parentheses, as in `WriteLine()`. The first introduction of a concept, such as *garbage collection* or *data encapsulation*, is

highlighted in italics. These conventions are intended to make the text more readable.

Acknowledgments

This book is the result of many invisible hands helping to keep its author on course. My most heartfelt thanks go to my wife, Beth, and my two children, Daniel and Anna. I have accumulated all too many IOUs in deferring this or that family outing in order to get this book done. Thank you all for being (mostly) patient and understanding and not too often asking if I was done yet.

I need to thank Caro Segal and Shimon Cohen of *you-niversity.com*, who provided me with a generous gift of time and encouragement. May the force be with you. I also owe a serious round of thanks to Eric Gunnerson, Peter Drayton, and Don Box, all of whom at one time or another fulfilled the role of white knight on horseback.

I would like to deeply thank Elena Driskill. Twice. First for the gift of those lovely drawings in Chapter 6. Second for her kind permission to reproduce them.

Deborah Lafferty has been my editor since the first edition of my *C++ Primer* back in 1986. She has been a constant source of good sense and understanding, and I deeply appreciate her encouragement (and prodding) in seeing this project through.

A pair of special production thanks go to Stephanie Hiebert and Steve Hall. Stephanie is the supreme copy editor of my nearly two decades of publishing. She made this a better book. Steve hoisted me back onto my typesetting saddle after having been thrown by wildly pernicious Framemaker problems. A tip of my virtual hat to the both of you.

The following reviewers offered numerous thoughtful comments and suggestions in reviewing various drafts of this manuscript: Indira Dhingra (special thanks for providing a final sanity check of the manuscript), Cay Horstmann, Eugene Kain, Jeff Kwak, Michael Lierheimer, Drew Nathanson, Clovis Tondo, and Damien Watkins.

Portions of this manuscript have been tried out in courses and talks held across the globe: Sydney, Amsterdam, Munich, Tel Aviv, Orlando, San Francisco, and San Jose. Thanks to everyone who provided feedback.

Resources

The richest documentation that you will be returning to time and again is the Visual Studio.NET documentation. The .NET framework reference is essential to doing any sort of C#/.NET programming.

Another rich source of information about .NET consists of the featured articles and columns in the *MSDN Magazine*. I'm always impressed by what I find in each issue. You can find it online at <http://msdn.microsoft.com/msdnmag>.

The DOTNET mailing list sponsored by DevelopMentor is a rich source of information. You can subscribe to it at <http://discuss.develop.com>.

Anything Jeffrey Richter, Don Box, Aaron Skonnard, or Jeff Prosise writes about .NET (or XML in Aaron's case) should be considered essential reading. Currently, most of their writing has appeared only as articles in *MSDN Magazine*.

Here is the collection of books that I have referenced or found helpful:

- *Active Server Pages+*, by Richard Anderson, Alex Homer, Rob Howard, and Dave Sussman, Wrox Press, Birmingham, England, 2000.
- *C# Essentials*, by Ben Albahari, Peter Drayton, and Brad Merrill, O'Reilly, Cambridge, MA, 2001.
- *C# Programming*, by Burton Harvey, Simon Robinson, Julian Templeman, and Karli Watson, Wrox Press, Birmingham, England, 2000.
- *Essential XML: Beyond Markup*, by Don Box, Aaron Skonnard, and John Lam, Addison-Wesley, Boston, 2000.
- *Microsoft C# Language Specifications*, Microsoft Press, Redmond, WA, 2001.
- *A Programmer's Introduction to C#*, 2nd Edition, by Eric Gunnerson, Apress, Berkeley, CA, 2001.

Stanley Lippman
Los Angeles
November 18, 2001
www.objectwrite.com

Contents

Preface	xiii
1 Hello, C#	1
1.1 A First C# Program	1
1.2 Namespaces	6
1.3 Alternative Forms of the <code>Main()</code> Function	10
1.4 Making a Statement	11
1.5 Opening a Text File for Reading and Writing	17
1.6 Formatting Output	19
1.7 The <code>string</code> Type	21
1.8 Local Objects	24
1.9 Value and Reference Types	28
1.10 The C# Array	29
1.11 The <code>new</code> Expression	30
1.12 Garbage Collection	32
1.13 Dynamic Arrays: The <code>ArrayList</code> Collection Class	33
1.14 The Unified Type System	35
1.14.1 <i>Shadow Boxing</i>	36
1.14.2 <i>Unboxing Leaves Us Downcast</i>	37
1.15 Jagged Arrays	39
1.16 The <code>Hashtable</code> Container	41
1.17 Exception Handling	44
1.18 A Basic Language Handbook for C#	47
1.18.1 <i>Keywords</i>	47
1.18.2 <i>Built-in Numeric Types</i>	49
1.18.3 <i>Arithmetic, Relational, and Conditional Operators</i>	51
1.18.4 <i>Operator Precedence</i>	54
1.18.5 <i>Statements</i>	55

2	Class Design	59
2.1	Our First Independent Class	59
2.2	Opening a New Visual Studio Project	63
2.3	Declaring Data Members	66
2.4	Properties	67
2.5	Indexers	69
2.6	Member Initialization	72
2.7	The Class Constructor	73
2.8	The implicit <code>this</code> Reference	76
2.9	<code>static</code> Class Members	79
2.10	<code>const</code> and <code>readonly</code> Data Members	81
2.11	The <code>enum</code> Value Type	83
2.12	The <code>delegate</code> Type	86
2.13	Function Parameter Semantics	92
2.13.1	Pass by Value	94
2.13.2	Pass by Reference: The <code>ref</code> Parameter	96
2.13.3	Pass by Reference: The <code>out</code> Parameter	97
2.14	Function Overloading	99
2.14.1	Resolving Overload Functions	100
2.14.2	Determining a Best Match	101
2.15	Variable-Length Parameter Lists	103
2.16	Operator Overloading	107
2.17	Conversion Operators	110
2.18	The Class Destructor	113
2.19	The <code>struct</code> Value Type	113
3	Object-Oriented Programming	117
3.1	Object-Oriented Programming Concepts	117
3.2	Supporting a Polymorphic Query Language	121
3.3	Designing a Class Hierarchy	124
3.4	Object Lessons	128
3.5	Designing an Abstract Base Class	132
3.6	Declaring an Abstract Base Class	133
3.7	Static Members of an Abstract Base Class	137
3.8	A Hybrid Abstract Base Class	138

3.8.1	<i>The Single-Inheritance Object Model</i>	140
3.8.2	<i>How Is a Hybrid Abstract Class Different?</i>	141
3.9	Defining a Derived Class	143
3.10	Overriding the Inherited Virtual Interface	145
3.11	Overriding the Virtual Object Methods	146
3.12	Member Access: The <code>new</code> and <code>base</code> Modifiers	147
3.12.1	<i>Accessibility versus Visibility</i>	150
3.12.2	<i>Encapsulating Base-Class Access</i>	151
3.13	Sealing a Class	153
3.14	The Exception Class Hierarchy	154
4	Interface Inheritance	159
4.1	Implementing a System Interface: <code>IComparable</code>	160
4.2	Accessing an Existing Interface	163
4.3	Defining an Interface	166
4.3.1	<i>Implementing Our Interface: Proof of Concept</i>	168
4.3.2	<i>Integrating Our Interface within the System Framework</i>	174
4.4	Explicit Interface Member Implementations	178
4.5	Inherited Interface Members	180
4.6	Overloaded, Hidden, or Ambiguous?	183
4.7	Mastering Copy Semantics: <code>ICloneable</code>	185
4.8	Mastering Finalize Semantics: <code>IDisposable</code>	187
4.9	<code>BitVector</code> : Extension through Composition	190
5	Exploring the System Namespace	199
5.1	Supporting the Fundamental Types	199
5.2	The Array Is a <code>System.Array</code>	200
5.3	Querying the Environment	203
5.3.1	<i>The Environment Class</i>	204
5.3.2	<i>Accessing All the Environment Variables</i>	205
5.3.3	<i>The Process Class</i>	207
5.3.4	<i>Finding the Logical Drives</i>	208
5.4	<code>System.IO</code>	209
5.4.1	<i>Handling File Extensions: The Path Class</i>	210
5.4.2	<i>Manipulating Directories</i>	212
5.4.3	<i>Manipulating Files</i>	215
5.4.4	<i>Reading and Writing Files</i>	216
5.5	A System Miscellany	221
5.5.1	<i>The <code>System.Collections.Stack</code> Container</i>	221

5.5.2	The <i>System.Diagnostics.TraceListener</i> Class	223
5.5.3	<i>System.Math</i>	225
5.5.4	The <i>DateTime</i> Class	226
5.6	Regular Expressions	228
5.7	<i>System.Threading</i>	235
5.8	The Web Request/Response Model	241
5.9	<i>System.Net.Sockets</i>	245
5.9.1	The Server-Side <i>TcpListener</i>	246
5.9.2	The Client-Side <i>TcpClient</i>	248
5.10	<i>System.Data</i>	249
5.10.1	The Database Tables	250
5.10.2	Opening the Database: Selecting a Data Provider	252
5.10.3	Navigating the <i>DataTable</i>	254
5.10.4	Setting Up the <i>DataRelation</i>	257
5.10.5	Selection and Expressions	258
5.11	<i>System.XML</i>	259
5.11.1	Getting XML Out of Our Programs	260
5.11.2	<i>XmlTextReader</i> : The Firehose	265
5.11.3	Document Object Model	272
5.11.4	<i>System.Xml.Xsl</i>	277
5.11.5	<i>System.Xml.XPath</i>	279
6	Windows Forms Designer	283
6.1	Our First Windows Forms Program	283
6.2	Building the GUI	285
6.3	Implementing the Event Callback Routines	288
6.3.1	Implementing a <i>TextBox</i> Event	292
6.3.2	Implementing the <i>Button</i> Events: OK	293
6.3.3	Implementing the <i>Button</i> Events: Quit	295
6.4	Inspecting and Generating Control Events	295
6.4.1	Labels Are Programmable	296
6.5	Implementing the <i>MessageBox</i> Pop-Up Dialog	298
6.6	The List Box for Unformatted Output	299
6.7	Exploring the File Dialog	302
6.8	A Pocketful of Buttons	304
6.9	Serving Up Menus	306
6.10	The <i>DataGrid</i> Control	308
6.11	Adding a <i>PictureBox</i> Control	310

7	ASP.NET and Web Forms Designer	315
7.1	Our First Web Forms Program	316
7.2	Opening an ASP.NET Web Application Project	316
7.2.1	<i>Modifying the Document Properties</i>	318
7.2.2	<i>Adding Controls to the Document: Label</i>	319
7.3	Adding Pages to a Project	320
7.4	The HyperLink Control: Linking to Other Pages	321
7.5	The DataGrid Control	321
7.6	Understanding the Page Event Life Cycle	323
7.7	The Data Provider	325
7.8	Web State Management	326
7.8.1	<i>Adding a TextBox Control</i>	328
7.8.2	<i>Adding an ImageButton Control</i>	329
7.8.3	<i>Adding a ListBox Control</i>	329
7.9	Managing State: Class Members	331
7.10	Managing State: The Session Object	332
7.11	Managing State: The Application Object	333
7.12	Validation Controls	334
7.13	Adding a DropDownList Control	335
7.14	Adding a Group of RadioButton Controls	337
7.15	Adding a CheckBoxList Control	338
7.16	Adding Validators to a Control	340
7.17	Adding a Calendar Control	344
7.18	Adding an Image Control	345
7.19	Programming Web Server Controls	345
8	The Common Language Runtime	349
8.1	Assemblies	349
8.2	Runtime Type Reflection	353
8.3	Modifying the Retrieval through BindingFlags	358
8.4	Invoking a Method during Runtime	362
8.5	Delegating the Test to Reflection	364
8.6	Attributes	367
8.6.1	<i>The Intrinsic Conditional Attribute</i>	367
8.6.2	<i>The Intrinsic Serializable Attribute</i>	369
8.6.3	<i>The Intrinsic Dllimport Attribute</i>	370
8.7	Implementing Our Own Attribute Class	372
8.7.1	<i>Positional and Named Parameters</i>	375

8.7.2 AttributeUsage	376
8.8 Runtime Discovery of Attributes Using Reflection	376
8.9 The Intermediate Language	378
8.9.1 Examining the Intermediate Language	379
8.9.2 The <code>ildasm</code> Tool	381
Index	385

Chapter 1

Hello, C#

My daughter has cycled through a number of musical instruments. With each one she is anxious to begin playing the classics—no, not Schubert or Schoenberg, but the Backstreet Boys and Britney Spears. Her various teachers, keen to keep her interest while grounding her in the fundamentals, have tended to indulge her. In a sense this chapter attempts the same precarious balance in presenting C#. In this context the classics are represented by Web Forms and Type Inheritance. The fundamentals are the seemingly mundane predefined language elements and mechanisms, such as scoping rules, arithmetic types, and namespaces. My approach is to introduce the language elements as they become necessary to implement a small first program. For those more traditionally minded, the chapter ends with a summary listing of the predefined language elements.

C# supports both integral and floating-point numeric types, as well as a Boolean type, a Unicode character type, and a high-precision decimal type. These are referred to as the *simple types*. Associated with these types is a set of operators, including addition (+), subtraction (-), equality (==), and inequality (!=). C# provides a predefined set of statements as well, such as the conditional `if` and `switch` statements and the looping `for`, `while`, and `foreach` statements. All of these, as well as the namespace and exception-handling mechanisms, are covered in this chapter.

1.1 A First C# Program

The traditional first program in a new language is one that prints *Hello, World!* on the user's console. In C# this program is implemented as follows:


```
// our first C# program
using System;
class Hello
{
    public static void Main()
    {
        Console.WriteLine( "Hello, World!" );
    }
}
```

When compiled and executed, this code generates the canonical

```
Hello, World!
```

Our program consists of four elements: (1) a comment, introduced by the double slash (`//`), (2) a `using` directive, (3) a class definition, and (4) a class member function (alternatively called a class method) named `Main()`.

A C# program begins execution in the class member function `Main()`. This is called the program entry point. `Main()` must be defined as `static`. In our example, we declare it as both `public` and `static`.

`public` identifies the level of access granted to `Main()`. A member of a class declared as `public` can be accessed from anywhere within the program. A class member is generally either a member function, performing a particular operation associated with the behavior of the class, or a data member, containing a value associated with the state of the class. Typically, class member functions are declared as `public` and data members are declared as `private`. (We'll look at member access levels again as we begin designing classes.)

Generally, the member functions of a class support the behavior associated with the class. For example, `WriteLine()` is a `public` member function of the `Console` class. `WriteLine()` prints its output to the user's console, followed by a new-line character. The `Console` class provides a `Write()` function as well. `Write()` prints its output to the terminal, but without inserting a new-line character. Typically, we use `Write()` when we wish the user to respond to a query posted to the console, and `WriteLine()` when we are simply displaying information. We'll see a relevant example shortly.

As C# programmers, our primary activity is the design and implementation of classes. What are classes? Usually they represent the entities in our applica-