原 版 风 暴 系 列

PEARSON
Addison Wesley

Effective Java

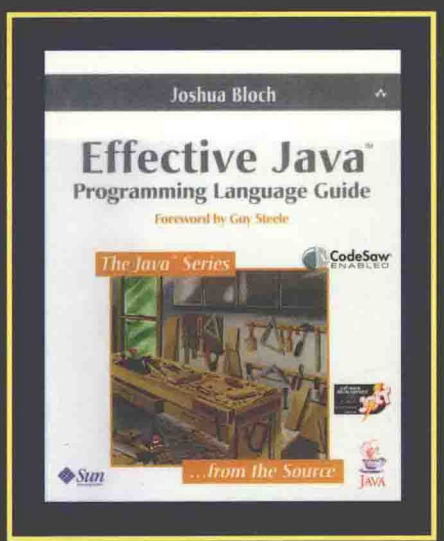
Programming Language Guide

（影印版）

2002 年度
Jolt 大奖

software development 2002 product excellence award JOLT

[美] Joshua Bloch 著



- 57 条极具实用价值的 Java 编程经验规则
- Java 之父 Gosling 大力推荐
- 涵盖开发人员每天所面临问题的解决方案

中国电力出版社
www.infopower.com.cn

原 版 风 暴 系 列

Effective Java

Programming Language Guide

（影印版）

[美] Joshua Bloch 著

中国电力出版社

Effective Java Programming Language Guide (ISBN 0-201-31005-8)
Joshua Bloch

北京市版权局著作合同登记号：图字：01-2003-7037
For sale and distribution in the People's Republic of China exclusively (except Taiwan, Hong Kong SAR and Macao SAR).
仅限于中华人民共和国境内（不包括中国香港、澳门特别行政区和中国台湾地区）销售发行。

图书在版编目（CIP）数据
Effective Java / （美）布洛克著. 影印本. —北京：中国电力出版社，2003
（原版风暴系列）
ISBN 7-5083-1813-7
Ⅰ.E... Ⅱ.布... Ⅲ.JAVA 语言—程序设计 Ⅳ.TP312
中国版本图书馆 CIP 数据核字（2003）第 092027 号

丛 书 名：原版风暴系列
书　　名：Effective Java（影印版）
编　　著：（美）Joshua Bloch
责任编辑：朱恩从
出版发行：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：16.75
书　　号：7-5083-1813-7
版　　次：2004年1月北京第1版　　2004年1月第1次印刷
定　　价：30.00 元

Effective Java™

Programming Language Guide

Joshua Bloch

To my family: Cindy, Tim, and Matt

Foreword

IF a colleague were to say to you, "Spouse of me this night today manufactures the unusual meal in a home. You will join?" three things would likely cross your mind: third, that you had been invited to dinner; second, that English was not your colleague's first language; and first, a good deal of puzzlement.

If you have ever studied a second language yourself and then tried to use it outside the classroom, you know that there are three things you must master: how the language is structured (grammar), how to name things you want to talk about (vocabulary), and the customary and effective ways to say everyday things (usage). Too often only the first two are covered in the classroom, and you find native speakers constantly suppressing their laughter as you try to make yourself understood.

It is much the same with a programming language. You need to understand the core language: is it algorithmic, functional, object-oriented? You need to know the vocabulary: what data structures, operations, and facilities are provided by the standard libraries? And you need to be familiar with the customary and effective ways to structure your code. Books about programming languages often cover only the first two, or discuss usage only spottily. Maybe that's because the first two are in some ways easier to write about. Grammar and vocabulary are properties of the language alone, but usage is characteristic of a community that uses it.

The Java programming language, for example, is object-oriented with single inheritance and supports an imperative (statement-oriented) coding style within each method. The libraries address graphic display support, networking, distributed computing, and security. But how is the language best put to use in practice?

There is another point. Programs, unlike spoken sentences and unlike most books and magazines, are likely to be changed over time. It's typically not enough to produce code that operates effectively and is readily understood by other persons; one must also organize the code so that it is easy to modify. There may be ten ways to write code for some task T. Of those ten ways, seven will be awkward, inefficient, or puzzling. Of the other three, which is most likely to be similar to the code needed for the task T' in next year's software release?

There are numerous books from which you can learn the grammar of the Java Programming Language, including *The Java Programming Language* by Arnold, Gosling, and Holmes [Arnold00] or *The Java Language Specification* by Gosling, Joy, yours truly, and Bracha [JLS]. Likewise, there are dozens of books on the libraries and APIs associated with the Java programming language.

This book addresses your third need: customary and effective usage. Joshua Bloch has spent years extending, implementing, and using the Java programming language at Sun Microsystems; he has also read a lot of other people's code, including mine. Here he offers good advice, systematically organized, on how to structure your code so that it works well, so that other people can understand it, so that future modifications and improvements are less likely to cause headaches—perhaps, even, so that your programs will be pleasant, elegant, and graceful.

Guy L. Steele Jr.
Burlington, Massachusetts
April 2001

Preface

In 1996 I pulled up stakes and headed west to work for JavaSoft, as it was then known, because it was clear that that was where the action was. In the intervening five years I've served as Java platform libraries architect. I've designed, implemented, and maintained many of the libraries and served as a consultant for many others. Presiding over these libraries as the Java platform matured was a once-in-a-lifetime opportunity. It is no exaggeration to say that I had the privilege to work with some of the great software engineers of our generation. In the process, I learned a lot about the Java programming language—what works, what doesn't, and how to use the language and its libraries to best effect.

This book is my attempt to share my experience with you so that you can imitate my successes while avoiding my failures. I borrowed the format from Scott Meyers's *Effective C++* [Meyers98], which consists of fifty items, each conveying one specific rule for improving your programs and designs. I found the format to be singularly effective, and I hope you do too.

In many cases, I took the liberty of illustrating the items with real-world examples from the Java platform libraries. When describing something that could have been done better, I tried to pick on code that I wrote myself, but occasionally I pick on something written by a colleague. I sincerely apologize if, despite my best efforts, I've offended anyone. Negative examples are cited not to cast blame but in the spirit of cooperation, so that all of us can benefit from the experience of those who've gone before.

While this book is not targeted solely at developers of reusable components, it is inevitably colored by my experience writing such components over the past two decades. I naturally think in terms of exported APIs (Application Programming Interfaces), and I encourage you to do likewise. Even if you aren't developing reusable components, thinking in these terms tends to improve the quality of the software you write. Furthermore, it's not uncommon to write a reusable component without knowing it: You write something useful, share it with your buddy across the hall, and before long you have half a dozen users. At this point, you no

longer have the flexibility to change the API at will and are thankful for all the effort that you put into designing the API when you first wrote the software.

My focus on API design may seem a bit unnatural to devotees of the new lightweight software development methodologies, such as *Extreme Programming* [Beck99]. These methodologies emphasize writing the simplest program that could possibly work. If you're using one of these methodologies, you'll find that a focus on API design serves you well in the *refactoring* process. The fundamental goals of refactoring are the improvement of system structure and the avoidance of code duplication. These goals are impossible to achieve in the absence of well-designed APIs for the components of the system.

No language is perfect, but some are excellent. I have found the Java programming language and its libraries to be immensely conducive to quality and productivity, and a joy to work with. I hope this book captures my enthusiasm and helps make your use of the language more effective and enjoyable.

Joshua Bloch
Cupertino, California
April 2001

Acknowledgments

I thank Patrick Chan for suggesting that I write this book and for pitching the idea to Lisa Friendly, the series managing editor; Tim Lindholm, the series technical editor; and Mike Hendrickson, executive editor of Addison-Wesley Professional. I thank Lisa, Tim, and Mike for encouraging me to pursue the project and for their superhuman patience and unyielding faith that I would someday write this book.

I thank James Gosling and his original team for giving me something great to write about, and I thank the many Java platform engineers who followed in James's footsteps. In particular, I thank my colleagues in Sun's Java Platform Tools and Libraries Group for their insights, their encouragement, and their support. The team consists of Andrew Bennett, Joe Darcy, Neal Gafter, Iris Garcia, Konstantin Kladko, Ian Little, Mike McCloskey, and Mark Reinhold. Former members include Zhenghua Li, Bill Maddox, and Naveen Sanjeeva.

I thank my manager, Andrew Bennett, and my director, Larry Abrahams, for lending their full and enthusiastic support to this project. I thank Rich Green, the VP of Engineering at Java Software, for providing an environment where engineers are free to think creatively and to publish their work.

I have been blessed with the best team of reviewers imaginable, and I give my sincerest thanks to each of them: Andrew Bennett, Cindy Bloch, Dan Bloch, Beth Bottos, Joe Bowbeer, Gilad Bracha, Mary Campione, Joe Darcy, David Eckhardt, Joe Fialli, Lisa Friendly, James Gosling, Peter Haggar, David Holmes, Brian Kernighan, Konstantin Kladko, Doug Lea, Zhenghua Li, Tim Lindholm, Mike McCloskey, Tim Peierls, Mark Reinhold, Ken Russell, Bill Shannon, Peter Stout, Phil Wadler, and two anonymous reviewers. They made numerous suggestions that led to great improvements in this book and saved me from many embarrassments. Any remaining embarrassments are my responsibility.

Numerous colleagues, inside and outside Sun, participated in technical discussions that improved the quality of this book. Among others, Ben Gomes, Steffen Grarup, Peter Kessler, Richard Roda, John Rose, and David Stoutamire contributed useful insights. A special thanks is due Doug Lea, who served as a

sounding board for many of the ideas in this book. Doug has been unfailingly generous with his time and his knowledge.

I thank Julie Dinicola, Jacqui Doucette, Mike Hendrickson, Heather Olszyk, Tracy Russ, and the whole team at Addison-Wesley for their support and professionalism. Even under an impossibly tight schedule, they were always friendly and accommodating.

I thank Guy Steele for writing the foreword. I am honored that he chose to participate in this project.

Finally, I thank my wife, Cindy Bloch, for encouraging and occasionally threatening me to write this book, for reading each item in its raw form, for helping me with Framemaker, for writing the index, and for putting up with me while I wrote.

Contents

CHAPTER 1

Introduction

THIS book is designed to help you make the most effective use of the Java™ programming language and its fundamental libraries, `java.lang`, `java.util`, and, to a lesser extent, `java.io`. The book discusses other libraries from time to time, but it does not cover graphical user interface programming or enterprise APIs.

This book consists of fifty-seven items, each of which conveys one rule. The rules capture practices generally held to be beneficial by the best and most experienced programmers. The items are loosely grouped into nine chapters, each concerning one broad aspect of software design. The book is not intended to be read from cover to cover: Each item stands on its own, more or less. The items are heavily cross-referenced so you can easily plot your own course through the book.

Most items are illustrated with program examples. A key feature of this book is that it contains code examples illustrating many *design patterns* and idioms. Some are old, like Singleton (Item 2), and others are new, like Finalizer Guardian (Item 6) and Defensive `readResolve` (Item 57). A separate index is provided for easy access to these patterns and idioms (page 239). Where appropriate, they are cross-referenced to the standard reference work in this area [Gamma95].

Many items contain one or more program examples illustrating some practice to be avoided. Such examples, sometimes known as *antipatterns*, are clearly labeled with a comment such as "`// Never do this!`" In each case, the item explains why the example is bad and suggests an alternative approach.

This book is not for beginners: it assumes that you are already comfortable with the Java programming language. If you are not, consider one of the many fine introductory texts [Arnold00, Campione00]. While the book is designed to be accessible to anyone with a working knowledge of the language, it should provide food for thought even for advanced programmers.

Most of the rules in this book derive from a few fundamental principles. Clarity and simplicity are of paramount importance. The user of a module should never be surprised by its behavior. Modules should be as small as possible but no

smaller. (As used in this book, the term *module* refers to any reusable software component, from an individual method to a complex system consisting of multiple packages.) Code should be reused rather than copied. The dependencies between modules should be kept to a minimum. Errors should be detected as soon as possible after they are made, ideally at compile time.

While the rules in this book do not apply 100 percent of the time, they do characterize best programming practices in the great majority of cases. You should not slavishly follow these rules, but you should violate them only occasionally and with good reason. Learning the art of programming, like most other disciplines, consists of first learning the rules and then learning when to violate them.

For the most part, this book is not about performance. It is about writing programs that are clear, correct, usable, robust, flexible, and maintainable. If you can do that, it's usually a relatively simple matter to get the performance you need (Item 37). Some items do discuss performance concerns, and a few of these items provide performance numbers. These numbers, which are introduced with the phrase "On my machine," should be regarded as approximate at best.

For what it's worth, my machine is an aging homebuilt 400 MHz Pentium® II with 128 megabytes of RAM, running Sun's 1.3 release of the Java 2 Standard Edition Software Development Kit (SDK) atop Microsoft Windows NT® 4.0. This SDK includes Sun's Java HotSpot™ Client VM, a state-of-the-art JVM implementation designed for client use.

When discussing features of the Java programming language and its libraries, it is sometimes necessary to refer to specific releases. For brevity, this book uses "engineering version numbers" in preference to official release names. Table 1.1 shows the correspondence between release names and engineering version numbers.

Table 1.1: Java Platform Versions

Official Release Name	Engineering Version Number
JDK 1.1.x / JRE 1.1.x	1.1
Java 2 Platform, Standard Edition, v 1.2	1.2
Java 2 Platform, Standard Edition, v 1.3	1.3
Java 2 Platform, Standard Edition, v 1.4	1.4

While features introduced in the 1.4 release are discussed in some items, program examples, with very few exceptions, refrain from using these features. The examples have been tested on release 1.3. Most, if not all, of them should run without modification on release 1.2.

The examples are reasonably complete, but they favor readability over completeness. They freely use classes from the packages `java.util` and `java.io`. In order to compile the examples, you may have to add one or both of these import statements:

```
import java.util.*;
import java.io.*;
```

Other boilerplate is similarly omitted. The book's Web site, `http://java.sun.com/docs/books/effective`, contains an expanded version of each example, which you can compile and run.

For the most part, this book uses technical terms as they are defined in *The Java Language Specification, Second Edition* [JLS]. A few terms deserve special mention. The language supports four kinds of types: *interfaces*, *classes*, *arrays*, and *primitives*. The first three are known as *reference types*. Class instances and arrays are *objects*; primitive values are not. A class's *members* consist of its *fields*, *methods*, *member classes*, and *member interfaces*. A method's *signature* consists of its name and the types of its formal parameters; the signature does *not* include the method's return type.

This book uses a few terms differently from the *The Java Language Specification*. Unlike *The Java Language Specification*, this book uses *inheritance* as a synonym for *subclassing*. Instead of using the term inheritance for interfaces, this book simply states that a class *implements* an interface or that one interface *extends* another. To describe the access level that applies when none is specified, this book uses the descriptive term *package-private* instead of the technically correct term *default access* [JLS, 6.6.1].

This book uses a few technical terms that are not defined in *The Java Language Specification*. The term *exported API*, or simply *API*, refers to the classes, interfaces, constructors, members, and serialized forms by which a programmer accesses a class, interface, or package. (The term *API*, which is short for *application programming interface*, is used in preference to the otherwise preferable term *interface* to avoid confusion with the language construct of that name.) A programmer who writes a program that uses an API is referred to as a *user* of the API. A class whose implementation uses an API is a *client* of the API.