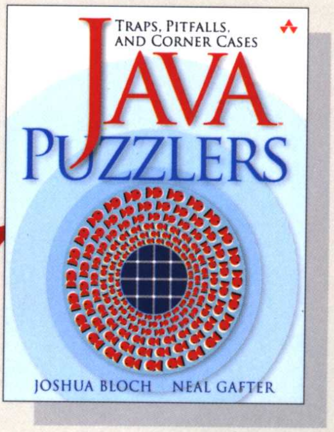
PEARSON
Addison
Wesley

典藏
原版书苑

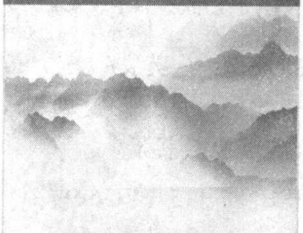
Java Puzzlers:
Traps, Pitfalls, and
Corner Cases

[美] Joshua Bloch Neal Gafter 著

# Java解惑
(英文版)



人民邮电出版社
POSTS & TELECOM PRESS



典藏原版书苑

# Java 解惑

（英文版）

Java Puzzlers：Traps，Pitfalls，and Corner Cases

Joshua Bloch
[美] 著
Neal Gafter

人民邮电出版社

**图书在版编目（CIP）数据**

Java 解惑. 英文/（美）布洛克（Bloch, J.），（美）加福特（Gafter, N.）著.
—北京：人民邮电出版社，2006.7
（典藏原版书苑）

ISBN 7-115-14956-9

Ⅰ. J…　Ⅱ. ①布…　②加…　Ⅲ. JAVA 语言—程序设计—英文　Ⅳ. TP312

中国版本图书馆 CIP 数据核字（2006）第 071838 号

典藏原版书苑

**Java 解惑（英文版）**

◆ 著　　[美] Joshua Bloch　Neal Gafter
　责任编辑　王　琳
◆ 人民邮电出版社出版发行　　北京市崇文区夕照寺街 14 号
　邮编　100061　电子函件　315@ptpress.com.cn
　网址　http://www.ptpress.com.cn
　北京顺义振华印刷厂印刷
　新华书店总店北京发行所经销
◆ 开本：800×1000　1/16
　印张：18.25
　字数：409 千字　　2006 年 7 月第 1 版
　印数：1 – 3 500 册　　2006 年 7 月北京第 1 次印刷

著作权合同登记号　图字：01-2006-3199 号

ISBN 7-115-14956-9/TP・5528

定价：32.00 元

**读者服务热线：(010)67132705　印装质量热线：(010)67129223**

# 前　言

与许多书一样，本书经历了长期的酝酿过程。我们收集 Java 谜题的时间与我们使用 Java 这种平台的时间一样长：如果你感兴趣的话，可以告诉你是从 1996 年中开始至今。在 2001 年初，我们产生了一个想法：搞一次完全由 Java 谜题所构成的演讲。我们把这个想法抛给了当时还在 Oracle 公司的 Larry Jacobs，他对这个想法非常支持。

2001 年 11 月在旧金山，我们在 Oracle Open World 会议上首次作了题为“Java 谜题”的演讲。为了增添魅力，我们介绍自己是“Type-it 兄弟，Click 和 Hack”，并且从 Tom 和 RayMagliozzi 主持的 Car Talk 节目中借用了一大堆的笑话。这个演讲被投票选为最佳演讲秀，即使我们不投自己的票结果可能也会如此。由此我们知道找对了路子。

从头到脚穿着蓝领工人那利索的制服，胸前装饰着 Java 的“咖啡杯”标志，我们在 JavaOne 2002 上再次利用在 Oracle 会议上的演讲来鼓吹我们的观点——至少我们的朋友是这么认为的。在接下来的年头里，我们又拿出了另外 3 个“Java 谜题”演讲，并且在数不胜数的会议、公司和大学里宣讲它们，足迹遍及全球许多城市，从奥斯陆到东京。这些演讲几乎得到了普遍的欢迎，几乎没人冲我们扔烂苹果。在 Linux Magazine 2003 年 3 月刊上，我们发表了一篇完全由 Java 谜题构成的文章，并且几乎没有收到任何厌恶我们的邮件。本书几乎包含了我们的演讲和文章中的所有谜题，以及许许多多其他的谜题。

尽管本书把注意力放到了 Java 平台的陷阱和缺陷上，但是我们并不是要以任何方式来诋毁 Java。因为热爱 Java，我们将近 10 年的职业生涯都奉献给了它。每一种具有强大能力的平台都会有某些问题，Java 与大多数平台相比问题已经少多了。你对问题理解得越透彻，你就越不会受到它们的影响，这正是本书要达到的目的。

本书中的多数谜题都是一些很短的程序，这些程序看起来在“明修栈道”，但实际却“暗渡陈仓”。这就是为什么我们选择视觉幻图来装饰本书的原因，这些幻图看起来是某样事物，但实际上却是另外一件东西。你在努力思考这些程序到底在做什么的时候，去盯着这些幻图好好看看。

毕竟，我们希望本书具有趣味性，真诚地希望你能够尽情享受解惑的乐趣，就像我们尽情享受编写它们的乐趣一样，还希望你能够从中学到很多东西，如我们曾经的那样。

不管怎样，请把你发现的谜题发给我们！如果你有一个你认为应该收录到本书将来的版本中的谜题，请把它写到一张 20 美元的账单后面，然后寄给我们，或者发 E-mail 到 puzzlers@javapuzzlers.com。如果采用了你发现的谜题，我们将向你付账。

最后要说的，但不是惟一要说的，就是请不要像我的兄弟那样编写代码。

# 内容提要

本书深入研究 Java 编程语言及其核心类库的细微之处，特写 95 个有关 Java 或其他类库的陷阱和缺陷的谜题，其中大多数谜题都采用短程序的形式给出。在每个谜题之后都有详细的解惑方案，这些方案在给出那些实际行为与表面上迥异的程序行为的简单解释的同时，更向读者展示了如何一劳永逸地避免底层的陷阱与缺陷。本书附录部分列出的陷阱及缺陷的目录，可供读者进一步学习参考。

本书以轻松诙谐的语言，寓教于乐的方式，由浅入深、总结归纳 Java 编程语言的知识点，适合具有 Java 知识的学习者和有编程经验的 Java 程序员阅读。

# ACKNOWLEDGMENTS

We thank the whole team at Addison-Wesley for their kindness and professional-ism. Early in the life of this project, Ann Sellers was our editor. Her infectiousenthusiasm helped get the project off to a good start. When Ann moved on, GregDoench, executive editor, took over. Greg is a wonderful editor and a perfect gen-tleman. He accommodated the many demands of this project without batting aneyelash. Greg's editorial assistant is Noreen Regina. Our project editor is TyrrellAlbaugh and our marketing manager is Stephane Nakib. Our cover designer isChuti Prasertsith and our copy editor is Evelyn Pyle. They all did great workunder a tight schedule.

We thank our management at Google for their support. Our director, PrabhaKrishna, was unfailingly encouraging. We thank Sergey Brin, Larry Page, andEric Schmidt for creating the best engineering environment on the planet.

We thank the many Java programmers who submitted bug reports to Sun overthe years, especially those who submitted bug reports that turned out not todescribe real bugs. Such bug reports were perhaps the richest source of puzzlermaterial: If correct behavior misled programmers into thinking that they had discovered a bug, it probably represents a trap or pitfall. In a similar vein, we thankSun for having the courage and wisdom to put the entire Java bug database on theWeb in 1996 [Bug]. This action was unheard of at the time; even today it is rare.

"Send us your puzzlers," we said at the end of each talk, and send them youdid—from all over the world. Special thanks are due Ron Gabor and Mike"madbot" McCloskey for the sheer magnitude of their contributions. Roncontributed Puzzles 28, 29, 30, and 31 and Mike contributed Puzzles 18, 23, 40,56, and 67. We thank Martin Buchholz for contributing Puzzle 81; ArmandDijamco for contributing Puzzle 14; Prof. Dr. Dominik Gruntz for contributingPuzzles 68 and 69; Kai Huang for contributing Puzzle 77; Jim Hugunin forcontributing Puzzle 45; Tim Huske for contributing Puzzle 41; Peter Kessler forcontributing Puzzle 35; Michael Klobe for contributing Puzzle 59; MagnusLundgren for contributing Puzzle 84; Scott Seligman for contributing Puzzle 22;Peter Stout for contributing Puzzle 39; Michael Tennes for contributing Puzzle 70;and Martin Traverso for contributing Puzzle 54.

We thank the dedicated band of reviewers who read the chapters of this bookin raw form: Peter von der Ahé, Pablo Belver, Tracy Bialik, Cindy Bloch, DanBloch, Beth Bottos, Joe Bowbeer, Joe Darcy, Bob Evans, Brian Goetz, Tim Hallo-ran, Barry Hayes, Tim Huske, Akiyoshi Kitaoka, Chris Lopez, Mike "madbot"McCloskey, Michael Nielsen, Tim Peierls, Peter Rathmann, Russ Rufer, SteveSchirripa, Yoshiki Shibata, Marshall Spight, Guy Steele, Dean Sutherland, MarkTaylor, Darlene Wallach, and Frank Yellin. They found flaws, suggested improve-ments, offered encouragement, and hurled invective. Any flaws that remain are thefault of my coauthor.

We thank the queen of the bloggers, Mary Smaragdis, for providing a homefor us on her celebrated blog [MaryBlog]. She graciously let us try out the mate-rial that became Puzzles 43, 53, 73, 87, and 94 on her readers. We judged the solu-tions and Mary gave out the prizes. For the record, the winners were Tom Hawtin,Tom Hawtin (again), Bob "Crazybob" Lee, Chris Nokleberg, and the mysteriousAT of Odessa, Ukraine. The discussions on Mary's blog contributed greatly tothese puzzles.

We thank our many supporters who responded enthusiastically to Java Puz-zlers over the years. The members of SDForum Java SIG served as guinea pigs foreach talk in its preliminary form. The JavaOne program committee provided ahome for the talks. Yuka Kamiya and Masayoshi Okutsu made the "Java Puzzlers"talks a success in Japan, where they took the form of real game shows withonstage contestants. Remarkably, the same person won every single contest: Theundisputed Java Puzzler champion of Japan is Yasuhiro Endoh.

We thank James Gosling and the many fine engineers who created the Javaplatform and improved it over the years. A book like this makes sense only for aplatform that is rock solid; without Java, there could be no "Java Puzzlers."

Numerous colleagues at Google, Sun, and elsewhere participated in technicaldiscussions that improved the quality of this book. Among others, Peter von derAhé, Dan Bloch, and Gilad Bracha contributed useful insights. We give specialthanks to Doug Lea, who served as a sounding board for many of the ideas in thebook. Once again, Doug was unfailingly generous with his time and knowledge.

We thank Professor Akiyoshi Kitaoka of the Department of Psychology atRitsumeikan University in Kyoto, Japan, for permission to use some of his opticalillusions to decorate this work. Professor Kitaoka's illusions are, quite simply,astonishing. Words cannot do them justice, so you owe it to yourself to take alook. He has two volumes available in Japanese [Kitaoka02, Kitaoka03]. AnEnglish translation encompassing both volumes is coming soon [Kitaoka05]. Inthe meantime, pay a visit to his Web site: http://www.ritsumei.ac.jp/~akitaoka/index-e.html. You won't be disappointed.

We thank Tom and Ray Magliozzi of *Car Talk* for providing jokes for us tosteal, and we thank their legal counsel of Dewey, Cheetham, and Howe for notsuing us.

We thank Josh's wife, Cindy, for helping us with FrameMaker, writing theindex, helping us edit the book, and designing the decorative stripe at the begin-ning of each chapter. Last but not least, we thank our families—Cindy, Tim, andMatt Bloch, and Ricki Lee, Sarah, and Hannah Gafter—for encouraging us towrite and for putting up with us while we wrote.

*Josh Bloch*
*Neal Gafter*
*San Jose, California*
*May 2005*

# Contents

# 1 Introduction

This book is filled with brainteasers about the Java programming language and its core libraries. Anyone with a working knowledge of Java can understand these puzzles, but many of them are tough enough to challenge even the most experienced programmer. Don't feel bad if you can't solve them. They are grouped loosely according to the features they use, but don't assume that the trick to a puzzle is related to its chapter heading; we reserve the right to mislead you.

Most of the puzzles exploit counterintuitive or obscure behaviors that can lead to bugs. These behaviors are known as *traps, pitfalls,* and *corner cases*. Every platform has them, but Java has far fewer than other platforms of comparable power. The goal of the book is to entertain you with puzzles while teaching you to avoid the underlying traps and pitfalls. By working through the puzzles, you will become less likely to fall prey to these dangers in your code and more likely to spot them in code that you are reviewing or revising.

This book is meant to be read with a computer at your side. To get the most out of the puzzles, you'll need a Java development environment, such as Sun's JDK [JDK-5.0]. It should support release 5.0, as some of the puzzles rely on features introduced in this release. You can download the source code for the puzzles from www.javapuzzlers.com. Unless you're a glutton for punishment, we recommend that you do this before solving the puzzles. It's a heck of a lot easier than typing them in yourself.

Most of the puzzles take the form of a short program that appears to do one thing but actually does something else. It's your job to figure out what the program does. To get the most out of these puzzles, we recommend that you take this approach:

1. Study the program and try to predict its behavior without using a computer. If you don't see a trick, keep looking.
2. Once you think you know what the program does, run it. Did it do what you thought it would? If not, can you come up with an explanation for the behavior you observed?
3. Think about how you might fix the program, assuming it is broken.
4. Then and only then, read the solution.

Some of the puzzles require you to write a small amount of code. To get the most out of these puzzles, we recommend that you try—at least briefly—to solve them without using a computer, and then test your solution on a computer. If your code doesn't work, play around with it and see whether you can make it work before reading the solution.

Unlike most puzzle books, this one alternates between puzzles and their solutions. This allows you to read the book without flipping back and forth between puzzles and solutions. The book is laid out so that you must turn the page to get from a puzzle to its solution, so you needn't fear reading a solution accidentally while you're still trying to solve a puzzle.

We encourage you to read each solution, even if you succeed in solving the puzzle. The solutions contain analysis that goes well beyond a simple explanation of the program's behavior. They discuss the relevant traps and pitfalls, and provide lessons on how to avoid falling prey to these hazards. Like most best-practice guidelines, these lessons are not hard-and-fast rules, but you should violate them only rarely and with good reason.

Most solutions contain references to relevant sections of *The Java™ Language Specification, Third Edition* [JLS]. These references aren't essential to understanding the puzzles, but they are useful if you want to delve deeper into the language rules underlying the puzzles. Similarly, many solutions contain references to relevant items in *Effective Java™ Programming Language Guide* [EJ]. These references are useful if you want to delve deeper into best practices.

Some solutions contain discussions of the language or API design decisions that led to the danger illustrated by the puzzle. These "lessons for language designers" are meant only as food for thought and, like other food, should be taken with a grain of salt. Language design decisions cannot be made in isolation. Every language embodies thousands of design decisions that interact in subtle ways. A design decision that is right for one language may be wrong for another.

Many of the traps and pitfalls in these puzzles are amenable to automatic detection by *static analysis*: analyzing programs without running them. Some excellent tools are available for detecting bugs by static analysis, such as Bill Pugh and David Hovemeyer's *FindBugs* [Hovemeyer04]. Some compilers and IDEs, such as Jikes and Eclipse, perform bug detection as well [Jikes, Eclipse]. If you are using one of these compilers, it is especially important that you not compile a puzzle until you've tried to solve it: The compiler's warning messages may give away the solution.

The appendix of this book is a catalog of the traps and pitfalls in the Java platform. It provides a concise taxonomy of the anomalies exploited by the puzzles, with references back to the puzzles and to other relevant resources. Do not look at the appendix until you're done solving the puzzles. Reading the appendix first would take all the fun out of the puzzles. After you've finished the puzzles, though, this is the place you'll turn to for reference.

# 2

# Expressive Puzzlers

The puzzles in this chapter are simple. They involve only expression evaluation. But remember, just because they're simple doesn't make them easy.

## Puzzle 1: Oddity

The following method purports to determine whether its sole argument is an odd number. Does the method work?

```
public static boolean isOdd(int i) {
    return i % 2 == 1;
}
```

## Solution 1: Oddity

An odd number can be defined as an integer that is divisible by 2 with a remainder of 1. The expression `i % 2` computes the remainder when `i` is divided by 2, so it would seem that this program ought to work. Unfortunately, it doesn't; it returns the wrong answer one quarter of the time.

Why one quarter? Because half of all `int` values are negative, and the `isOdd` method fails for all negative odd values. It returns `false` when invoked on any negative value, whether even or odd.

This is a consequence of the definition of Java's remainder operator (%). It is defined to satisfy the following identity for all `int` values `a` and all nonzero `int` values `b`:

```
(a / b) * b + (a % b) == a
```

In other words, if you divide `a` by `b`, multiply the result by `b`, and add the remainder, you are back where you started [JLS 15.17.3]. This identity makes perfect sense, but in combination with Java's truncating integer division operator [JLS 15.17.2], it implies that **when the remainder operation returns a nonzero result, it has the same sign as its left operand.**

The `isOdd` method and the definition of the term *odd* on which it was based both assume that all remainders are positive. Although this assumption makes sense for some kinds of division [Boxing], Java's remainder operation is perfectly matched to its integer division operation, which discards the fractional part of its result.

When `i` is a negative odd number, `i % 2` is equal to `-1` rather than `1`, so the `isOdd` method incorrectly returns `false`. To prevent this sort of surprise, **test that your methods behave properly when passed negative, zero, and positive values for each numerical parameter.**

The problem is easy to fix. Simply compare `i % 2` to `0` rather than to `1`, and reverse the sense of the comparison:

```
public static boolean isOdd(int i) {
    return i % 2 != 0;
}
```