

Joshua Bloch

Updated
for
Java 9



Effective Java

Third Edition

Best practices for



...the Java Platform



About This E-Book

EPUB is an open, industry-standard format for e-books. However, support for EPUB and its many features varies across reading devices and applications. Use your device or app settings to customize the presentation to your liking. Settings that you can customize often include font, font size, single or double column, landscape or portrait mode, and figures that you can click or tap to enlarge. For additional information about the settings and features on your reading device or app, visit the device manufacturer's Web site.

Many titles include programming code or configuration examples. To optimize the presentation of these elements, view the e-book in single-column, landscape mode and adjust the font size to the smallest setting. In addition to presenting code and configurations in the reflowable text format, we have included images of the code that mimic the presentation found in the print book; therefore, where the reflowable format may compromise the presentation of the code listing, you will see a “Click here to view code image” link. Click the link to view the print-fidelity code image. To return to the previous page viewed, click the Back button on your device or app.

Effective Java

Third Edition

Joshua Bloch

★Addison-Wesley

Boston • Columbus • Indianapolis • New York •
San Francisco • Amsterdam • Cape Town
Dubai • London • Madrid • Milan • Munich • Paris
• Montreal • Toronto • Delhi • Mexico City
São Paulo • Sydney • Hong Kong • Seoul •
Singapore • Taipei • Tokyo

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection

with or arising out of the use of the information or programs contained herein.

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact intlcs@pearson.com.

Visit us on the Web: informit.com/aw

Library of Congress Control Number: 2017956176

Copyright © 2018 Pearson Education Inc.

Portions copyright © 2001-2008 Oracle and/or its affiliates.

All Rights Reserved.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, request forms and the appropriate contacts within the Pearson Education Global Rights & Permissions Department, please visit www.pearsoned.com/permissions/.

ISBN-13: 978-0-13-468599-1

ISBN-10: 0-13-468599-7

To my family: Cindy, Tim, and Matt

Contents

Foreword

Preface

Acknowledgments

1 Introduction

2 Creating and Destroying Objects

Item 1: Consider static factory methods instead of constructors

Item 2: Consider a builder when faced with many constructor parameters

Item 3: Enforce the singleton property with a private constructor or an enum type

Item 4: Enforce noninstantiability with a private constructor

Item 5: Prefer dependency injection to hardwiring resources

Item 6: Avoid creating unnecessary objects

Item 7: Eliminate obsolete object references

Item 8: Avoid finalizers and cleaners

Item 9: Prefer `try-with-resources` to `try-finally`

3 Methods Common to All Objects

Item 10: Obey the general contract when overriding `equals`

Item 11: Always override `hashCode` when you override `equals`

Item 12: Always override `toString`

Item 13: Override `clone` judiciously

Item 14: Consider implementing `Comparable`

4 Classes and Interfaces

Item 15: Minimize the accessibility of classes and members

Item 16: In public classes, use accessor methods, not public fields

Item 17: Minimize mutability

Item 18: Favor composition over inheritance

Item 19: Design and document for inheritance or else prohibit it

Item 20: Prefer interfaces to abstract classes

Item 21: Design interfaces for posterity

Item 22: Use interfaces only to define types

Item 23: Prefer class hierarchies to tagged classes

Item 24: Favor static member classes over nonstatic

Item 25: Limit source files to a single top-level class

5 Generics

Item 26: Don't use raw types

Item 27: Eliminate unchecked warnings

Item 28: Prefer lists to arrays

Item 29: Favor generic types

Item 30: Favor generic methods

Item 31: Use bounded wildcards to increase API flexibility

Item 32: Combine generics and varargs judiciously

Item 33: Consider typesafe heterogeneous containers

6 Enums and Annotations

Item 34: Use enums instead of `int` constants

Item 35: Use instance fields instead of ordinals

Item 36: Use `EnumSet` instead of bit fields

Item 37: Use `EnumMap` instead of ordinal indexing

Item 38: Emulate extensible enums with interfaces

Item 39: Prefer annotations to naming patterns

Item 40: Consistently use the `Override` annotation

Item 41: Use marker interfaces to define types

7 Lambdas and Streams

Item 42: Prefer lambdas to anonymous classes

Item 43: Prefer method references to lambdas

Item 44: Favor the use of standard functional interfaces

Item 45: Use streams judiciously

Item 46: Prefer side-effect-free functions in streams

Item 47: Prefer `Collection` to `Stream` as a return type

Item 48: Use caution when making streams parallel

8 Methods

Item 49: Check parameters for validity

Item 50: Make defensive copies when needed

Item 51: Design method signatures carefully

Item 52: Use overloading judiciously

Item 53: Use `varargs` judiciously

Item 54: Return empty collections or arrays, not `nulls`

Item 55: Return optionals judiciously

Item 56: Write doc comments for all exposed API elements

9 General Programming

Item 57: Minimize the scope of local variables

Item 58: Prefer for-each loops to traditional `for` loops

Item 59: Know and use the libraries

Item 60: Avoid `float` and `double` if exact answers are required

Item 61: Prefer primitive types to boxed primitives

Item 62: Avoid strings where other types are more appropriate

Item 63: Beware the performance of string concatenation

Item 64: Refer to objects by their interfaces

Item 65: Prefer interfaces to reflection

Item 66: Use native methods judiciously

Item 67: Optimize judiciously

Item 68: Adhere to generally accepted naming conventions

10 Exceptions

Item 69: Use exceptions only for exceptional conditions

Item 70: Use checked exceptions for recoverable conditions and runtime exceptions for programming errors

Item 71: Avoid unnecessary use of checked exceptions

Item 72: Favor the use of standard exceptions

Item 73: Throw exceptions appropriate to the abstraction

Item 74: Document all exceptions thrown by each method

Item 75: Include failure-capture information in detail messages

Item 76: Strive for failure atomicity

Item 77: Don't ignore exceptions

11 Concurrency

Item 78: Synchronize access to shared mutable data

Item 79: Avoid excessive synchronization

Item 80: Prefer executors, tasks, and streams to threads

Item 81: Prefer concurrency utilities to `wait` and `notify`

Item 82: Document thread safety

Item 83: Use lazy initialization judiciously

Item 84: Don't depend on the thread scheduler

12 Serialization

Item 85: Prefer alternatives to Java serialization

Item 86: Implement `Serializable` with great caution

Item 87: Consider using a custom serialized form

Item 88: Write `readObject` methods defensively

Item 89: For instance control, prefer enum types to `readResolve`

Item 90: Consider serialization proxies instead of serialized instances

Items Corresponding to Second Edition

References

Index

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, or *The Java™ Language Specification* by Gosling, Joy, yours truly,

and Bracha. 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

PREFACE TO THE THIRD EDITION

IN 1997, when Java was new, James Gosling (the father of Java), described it as a “blue collar language” that was “pretty simple” [Gosling97]. At about the same time, Bjarne Stroustrup (the father

of C++) described C++ as a “multi-paradigm language” that “deliberately differs from languages designed to support a single way of writing programs” [Stroustrup95]. Stroustrup warned:

Much of the relative simplicity of Java is—like for most new languages—partly an illusion and partly a function of its incompleteness. As time passes, Java will grow significantly in size and complexity. It will double or triple in size and grow implementation-dependent extensions or libraries. [Stroustrup]

Now, twenty years later, it’s fair to say that Gosling and Stroustrup were both right. Java is now large and complex, with multiple abstractions for many things, from parallel execution, to iteration, to the representation of dates and times.

I still like Java, though my ardor has cooled a bit as the platform has grown. Given its increased size and complexity, the need for an up-to-date best-practices guide is all the more critical. With this third edition of *Effective Java*, I did my best to provide you with one. I hope this edition continues to satisfy the need, while staying true to the spirit of the first two editions.

Small is beautiful, but simple ain’t easy.

San Jose, California

November 2017

P.S. I would be remiss if I failed to mention an industry-wide best practice that has occupied a fair amount of my time lately. Since the birth of our field in the 1950’s, we have freely reimplemented each others’ APIs. This practice was critical to the meteoric success of computer technology. I am active in the effort to preserve this freedom [CompSci17], and I encourage you to join me. It is crucial to the continued health of our profession that we retain the right to reimplement each others’ APIs.