The Java® Virtual Machine Specification

Java SE 8 Edition

Tim Lindholm, Frank Yellin, Gilad Bracha, Alex Buckley

ORACLE

The Java® Virtual Machine Specification

Java SE 8 Edition

This page intentionally left blank

The Java® Virtual Machine Specification

Java SE 8 Edition

Tim Lindholm
Frank Yellin
Gilad Bracha
Alex Buckley

Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact U.S. Corporate and Government Sales, (800) 382-3419, `corpsales@pearsontechgroup.com`. For sales outside the United States, please contact International Sales, `international@pearson.com`.

Visit us on the Web: `informit.com/aw`

Library of Congress Control Number: 2014936247
ISBN-13: 978-0-13-390590-8
ISBN-10: 0-13-390590-X

Text printed in the United States on recycled paper at Edwards Brothers Malloy in Ann Arbor, Michigan. First printing, May 2014.

To Sophia and Susan, in deepest appreciation.

This page intentionally left blank

Table of Contents

Preface to the Java SE 8 Edition

THE Java SE 8 Edition of *The Java® Virtual Machine Specification* incorporates all the changes that have been made to the Java Virtual Machine since the Java SE 7 Edition in 2011. In addition, numerous corrections and clarifications have been made to align with popular implementations of the Java Virtual Machine.

This Edition continues the tradition of specifying the *abstract* Java Virtual Machine, serving as documentation for a concrete implementation only as a blueprint documents a house. An implementation of the Java Virtual Machine must embody this specification, but is constrained by it only where absolutely necessary.

Notable changes to the Java programming language in Java SE 8 have brought corresponding changes to the Java Virtual Machine. To maximize binary compatibility, it has been desirable to specify default methods directly in the Java Virtual Machine, rather than relying on compiler magic that might not be portable across vendors or product releases, and is certainly not applicable to pre-existing `class` files. In the context of JSR 335, *Lambda Expressions for the Java Programming Language*, Dan Smith at Oracle consulted with implementers to determine how best to integrate default methods into the constant pool and method structures, the method and interface method resolution algorithms, and the bytecode instruction set. JSR 335 also introduced `private` and `static` methods in interfaces at the `class` file level; they too have been carefully integrated with interface method resolution.

A theme of Java SE 8 is co-evolution of the Java SE platform libraries with the Java Virtual Machine. A small but useful example is support for method parameter names at run time: storing such names in the `class` file structure goes hand in hand with offering a standard API to retrieve them (`java.lang.reflect.Parameter`). This illustrates an interesting development in the `class` file structure over the years: the First Edition of this specification defined six attributes, of which three were deemed critical to the Java Virtual Machine, while this Java SE 8 Edition defines 23 attributes, of which five are deemed critical to the Java Virtual Machine; that is to say, attributes now exist primarily to support libraries and tools rather than the Java Virtual Machine itself. To help readers understand the `class` file structure, this specification more clearly documents the role of each attribute and the constraints placed upon it.

Many colleagues in the Java Platform Group at Oracle have provided valuable support to this specification: Mandy Chung, Joe Darcy, Joel Franck, Staffan Friberg, Yuri Gaevsky, Jon Gibbons, Jeannette Hung, Eric McCorkle, Matherey Nunez, Mark Reinhold, John Rose, Georges Saab, Steve Sides, Bernard Traversat, Michel Trudeau, and Mikael Vidstedt. Particular thanks to Dan Heidinga (IBM), Karen Kinnear, Keith McGuigan, and Harold Seigel for their ironclad commitment to compatibility and security in popular Java Virtual Machine implementations.

Alex Buckley
Santa Clara, California

March, 2014

CHAPTER 1

Introduction

1.1 A Bit of History

The Java® programming language is a general-purpose, concurrent, object-oriented language. Its syntax is similar to C and C++, but it omits many of the features that make C and C++ complex, confusing, and unsafe. The Java platform was initially developed to address the problems of building software for networked consumer devices. It was designed to support multiple host architectures and to allow secure delivery of software components. To meet these requirements, compiled code had to survive transport across networks, operate on any client, and assure the client that it was safe to run.

The popularization of the World Wide Web made these attributes much more interesting. Web browsers enabled millions of people to surf the Net and access media-rich content in simple ways. At last there was a medium where what you saw and heard was essentially the same regardless of the machine you were using and whether it was connected to a fast network or a slow modem.

Web enthusiasts soon discovered that the content supported by the Web's HTML document format was too limited. HTML extensions, such as forms, only highlighted those limitations, while making it clear that no browser could include all the features users wanted. Extensibility was the answer.

The HotJava browser first showcased the interesting properties of the Java programming language and platform by making it possible to embed programs inside HTML pages. Programs are transparently downloaded into the browser along with the HTML pages in which they appear. Before being accepted by the browser, programs are carefully checked to make sure they are safe. Like HTML pages, compiled programs are network- and host-independent. The programs behave the same way regardless of where they come from or what kind of machine they are being loaded into and run on.

A Web browser incorporating the Java platform is no longer limited to a predetermined set of capabilities. Visitors to Web pages incorporating dynamic content can be assured that their machines cannot be damaged by that content. Programmers can write a program once, and it will run on any machine supplying a Java run-time environment.

1.2 The Java Virtual Machine

The Java Virtual Machine is the cornerstone of the Java platform. It is the component of the technology responsible for its hardware- and operating system-independence, the small size of its compiled code, and its ability to protect users from malicious programs.

The Java Virtual Machine is an abstract computing machine. Like a real computing machine, it has an instruction set and manipulates various memory areas at run time. It is reasonably common to implement a programming language using a virtual machine; the best-known virtual machine may be the P-Code machine of UCSD Pascal.

The first prototype implementation of the Java Virtual Machine, done at Sun Microsystems, Inc., emulated the Java Virtual Machine instruction set in software hosted by a handheld device that resembled a contemporary Personal Digital Assistant (PDA). Oracle's current implementations emulate the Java Virtual Machine on mobile, desktop and server devices, but the Java Virtual Machine does not assume any particular implementation technology, host hardware, or host operating system. It is not inherently interpreted, but can just as well be implemented by compiling its instruction set to that of a silicon CPU. It may also be implemented in microcode or directly in silicon.

The Java Virtual Machine knows nothing of the Java programming language, only of a particular binary format, the `class` file format. A `class` file contains Java Virtual Machine instructions (or *bytecodes*) and a symbol table, as well as other ancillary information.

For the sake of security, the Java Virtual Machine imposes strong syntactic and structural constraints on the code in a `class` file. However, any language with functionality that can be expressed in terms of a valid `class` file can be hosted by the Java Virtual Machine. Attracted by a generally available, machine-independent platform, implementors of other languages can turn to the Java Virtual Machine as a delivery vehicle for their languages.

The Java Virtual Machine specified here is compatible with the Java SE 8 platform, and supports the Java programming language specified in *The Java Language Specification, Java SE 8 Edition*.

1.3 Organization of the Specification

Chapter 2 gives an overview of the Java Virtual Machine architecture.

Chapter 3 introduces compilation of code written in the Java programming language into the instruction set of the Java Virtual Machine.

Chapter 4 specifies the `class` file format, the hardware- and operating system-independent binary format used to represent compiled classes and interfaces.

Chapter 5 specifies the start-up of the Java Virtual Machine and the loading, linking, and initialization of classes and interfaces.

Chapter 6 specifies the instruction set of the Java Virtual Machine, presenting the instructions in alphabetical order of opcode mnemonics.

Chapter 7 gives a table of Java Virtual Machine opcode mnemonics indexed by opcode value.

In the Second Edition of *The Java® Virtual Machine Specification*, Chapter 2 gave an overview of the Java programming language that was intended to support the specification of the Java Virtual Machine but was not itself a part of the specification. In *The Java Virtual Machine Specification, Java SE 8 Edition*, the reader is referred to *The Java Language Specification, Java SE 8 Edition* for information about the Java programming language. References of the form: (JLS §x.y) indicate where this is necessary.

In the Second Edition of *The Java® Virtual Machine Specification*, Chapter 8 detailed the low-level actions that explained the interaction of Java Virtual Machine threads with a shared main memory. In *The Java Virtual Machine Specification, Java SE 8 Edition*, the reader is referred to Chapter 17 of *The Java Language Specification, Java SE 8 Edition* for information about threads and locks. Chapter 17 reflects *The Java Memory Model and Thread Specification* produced by the JSR 133 Expert Group.

1.4 Notation

Throughout this specification we refer to classes and interfaces drawn from the Java SE platform API. Whenever we refer to a class or interface (other than those declared in an example) using a single identifier *N*, the intended reference is to the class or interface named *N* in the package `java.lang`. We use the fully qualified name for classes or interfaces from packages other than `java.lang`.

Whenever we refer to a class or interface that is declared in the package `java` or any of its subpackages, the intended reference is to that class or interface as loaded by the bootstrap class loader (§5.3.1).

Whenever we refer to a subpackage of a package named `java`, the intended reference is to that subpackage as determined by the bootstrap class loader.

The use of fonts in this specification is as follows:

- A `fixed width` font is used for Java Virtual Machine data types, exceptions, errors, `class` file structures, Prolog code, and Java code fragments.
- *Italic* is used for Java Virtual Machine "assembly language", its opcodes and operands, as well as items in the Java Virtual Machine's run-time data areas. It is also used to introduce new terms and simply for emphasis.

Non-normative information, designed to clarify the specification, is given in smaller, indented text.

> This is non-normative information. It provides intuition, rationale, advice, examples, etc.

1.5 Feedback

Readers may send feedback about errors, omissions, and ambiguities in this specification to `jvms-comments_ww@oracle.com`.

Questions concerning the generation and manipulation of `class` files by `javac` (the reference compiler for the Java programming language) may be sent to `compiler-dev@openjdk.java.net`.

Index

A

B

C

D

E

F

J

L

M

N

O

P

R

S

T

U

V

W