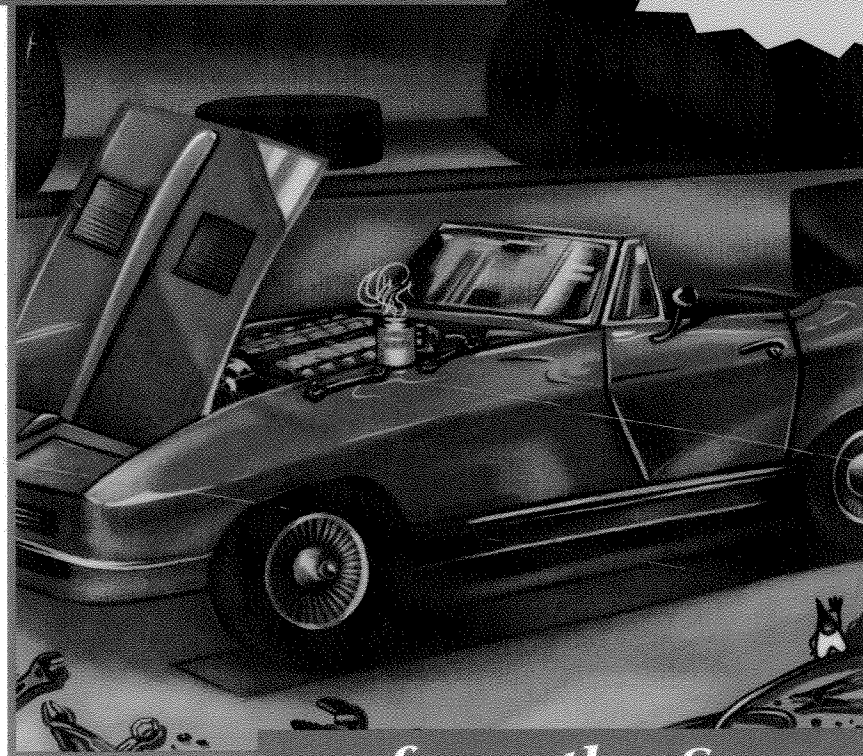


Tim Lindholm • Frank Yellin

The Java™ Virtual Machine Specification Second Edition

The Java Series

Java™ 2 Platform



... from the Source™



Juniper Ex. 1017-p. 1
Juniper v Finjan



*The JavaTM
Virtual Machine
Specification
Second Edition*

**Tim Lindholm
Frank Yellin**



ADDISON-WESLEY

An imprint of Addison Wesley Longman, Inc.

Reading, Massachusetts • Harlow, England • Menlo Park, California
Berkeley, California • Don Mills, Ontario • Sydney
Bonn • Amsterdam • Tokyo • Mexico City

Copyright © 1997-1999 Sun Microsystems, Inc.
901 San Antonio Road, Palo Alto, California 94303 U.S.A.
All rights reserved.

Duke™ designed by Joe Palrang.

RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the United States Government is subject to the restrictions set forth in DFARS 252.227-7013 (c)(1)(ii) and FAR 52.227-19.

The release described in this manual may be protected by one or more U.S. patents, foreign patents, or pending applications.

Sun Microsystems, Inc. (SUN) hereby grants to you a fully paid, nonexclusive, nontransferable, perpetual, worldwide limited license (without the right to sublicense) under SUN's intellectual property rights that are essential to practice this specification. This license allows and is limited to the creation and distribution of clean room implementations of this specification that: (i) include a complete implementation of the current version of this specification without subsetting or supersetting; (ii) implement all the interfaces and functionality of the required packages of the Java® 2 Platform, Standard Edition, as defined by SUN, without subsetting or supersetting; (iii) do not add any additional packages, classes, or interfaces to the java.* or javax.* packages or their subpackages; (iv) pass all test suites relating to the most recent published version of the specification of the Java® 2 Platform, Standard Edition, that are available from SUN six (6) months prior to any beta release of the clean room implementation or upgrade thereto; (v) do not derive from SUN source code or binary materials; and (vi) do not include any SUN source code or binary materials without an appropriate and separate license from SUN.

Sun, Sun Microsystems, Sun Microsystems Computer Corporation, the Sun logo, the Sun Microsystems Computer Corporation logo, Solaris, Java, JavaSoft, JavaScript, HotJava, JDK, and all Java-based trademarks or logos are trademarks or registered trademarks of Sun Microsystems, Inc. UNIX® is a registered trademark in the United States and other countries, exclusively licensed through X/Open Company, Ltd. All other product names mentioned herein are the trademarks of their respective owners.

THIS PUBLICATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT.

THIS PUBLICATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THE PUBLICATION. SUN MICROSYSTEMS, INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS PUBLICATION AT ANY TIME.

Library of Congress Cataloging-in-Publication Data

Lindholm, Tim

The Java virtual machine specification / Tim Lindholm, Frank
Yellin. -- 2nd ed.

p. cm.

Includes bibliographical references and index.

ISBN 0-201-43294-3

1. Java (Computer program language) 2. Internet (Computer
network) 3. Virtual computer systems. I. Yellin, Frank.

II. Title.

QA76.73.J38L56 1999

005.13'3--dc21

99-18470

CIP

Text printed on recycled and acid-free paper

1 2 3 4 5 6 7 8 9 -MA- 03 02 01 00 99

First printing, April 1999

The element type of an array may be any type, whether primitive or reference. In particular, arrays with an interface type as the component type are supported; the elements of such an array may have as their value a null reference or instances of any class type that implements the interface. Arrays with an abstract class type as the component type are supported; the elements of such an array may have as their value a null reference or instances of any subclass of this abstract class that is not itself abstract.

2.15.2 Array Variables

A variable of array type holds a reference to an object. Declaring a variable of array type does not create an array object or allocate any space for array components. It creates only the variable itself, which can contain a reference to an array.

Because an array's length is not part of its type, a single variable of array type may contain references to arrays of different lengths. Once an array object is created, its length never changes. To make an array variable refer to an array of different length, a reference to a different array must be assigned to the variable.

If an array variable v has type $A[]$, where A is a reference type, then v can hold a reference to any array type $B[]$, provided B can be assigned to A (§2.6.7).

2.15.3 Array Creation

An array is created by an *array creation expression* or an *array initializer*.

2.15.4 Array Access

A component of an array is accessed using an *array access expression*. Arrays may be indexed by `int` values; `short`, `byte`, or `char` values may also be used as they are subjected to unary numeric promotion (§2.6.10) and become `int` values.

All arrays are 0-origin. An array with length n can be indexed by the integers 0 through $n - 1$. All array accesses are checked at run time; an attempt to use an index that is less than zero or greater than or equal to the length of the array causes an `ArrayIndexOutOfBoundsException` to be thrown.

2.16 Exceptions

When a program violates the semantic constraints of the Java programming language, the Java virtual machine signals this error to the program as an *exception*. An

example of such a violation is an attempt to index outside the bounds of an array. The Java programming language specifies that an exception will be thrown when semantic constraints are violated and will cause a nonlocal transfer of control from the point where the exception occurred to a point that can be specified by the programmer. An exception is said to be *thrown* from the point where it occurred and is said to be *caught* at the point to which control is transferred. A method invocation that completes because an exception causes transfer of control to a point outside the method is said to *complete abruptly*.

Programs can also throw exceptions explicitly, using `throw` statements. This provides an alternative to the old-fashioned style of handling error conditions by returning distinguished error values, such as the integer value `-1`, where a negative value would not normally be expected.

Every exception is represented by an instance of the class `Throwable` or one of its subclasses; such an object can be used to carry information from the point at which an exception occurs to the handler that catches it. Handlers are established by `catch` clauses of `try` statements. During the process of throwing an exception, the Java virtual machine abruptly completes, one by one, any expressions, statements, method and constructor invocations, static initializers, and field initialization expressions that have begun but not completed execution in the current thread. This process continues until a handler is found that indicates that it handles the thrown exception by naming the class of the exception or a superclass of the class of the exception. If no such handler is found, then the method `uncaughtException` is invoked for the `ThreadGroup` that is the parent of the current thread.

In the Java programming language the exception mechanism is integrated with the synchronization model (§2.19) so that locks are properly released as synchronized statements and so that invocations of synchronized methods complete abruptly.

The specific exceptions covered in this section are that subset of the predefined exceptions that can be thrown directly by the operation of the Java virtual machine. Additional exceptions can be thrown by class library or user code; these exceptions are not covered here. See *The Java™ Language Specification* for information on all predefined exceptions.

2.16.1 The Causes of Exceptions

An exception is thrown for one of three reasons:

1. An abnormal termination of the program, such as a hardware error or a power failure, causes the JVM to abruptly complete the execution of the program.
 - When the JVM abruptly completes the execution of the program, it throws an `OutOfMemoryError`.
 - When the JVM abruptly completes the execution of the program, it throws an `InternalError`.
 - When the JVM abruptly completes the execution of the program, it throws an `InternalError`.
2. A thread terminates abruptly because it has completed its execution.
3. An exception is thrown.
 - The JVM throws an exception when it detects a violation of the Java language rules.
 - An exception is thrown when a program attempts to perform an illegal operation.

Exceptions are represented by instances of the class `Throwable` or one of its subclasses.

2.16.2 Handling Exceptions

When an exception is thrown, the JVM handles the exception by performing the following steps:

A stack of frames is created for the exception. The stack appears as a sequence of frames, each of which contains a reference to the catch clause that caught the exception.

The JVM searches the stack for a handler for the exception.

- If a handler is found, the JVM transfers control to the handler.
- If no handler is found, the JVM terminates the program.
- If a handler is found, the JVM transfers control to the handler.
- If no handler is found, the JVM terminates the program.