Includes Companion Web Site

The JAVA™ Developer's Toolkit

Techniques and Technologies for Web Programmers

- Learn next-generation Java techniques
- Input/output, including graphics and sound
- Java multithreading and networking
- Java interaction with RMI, OLE, ODBC, and native methods

JOSHUA MARKETOS

The Java™ Developer's Toolkit

Joshua Marketos

WILEY COMPUTER PUBLISHING

John Wiley & Sons, Inc.

New York • Chichester • Brisbane • Toronto • Singapore • Weinheim

Library of Congress Cataloging-in-Publication Data:

Marketos, Joshua.
The Java Developer's Toolkit : Techniques and Technologies for Experienced Web Programmers / Joshua Marketos.
p. 400 cm.
Includes index.
ISBN 0-471-16519-0 (pbk. : alk. paper)
1. Java (Computer program language). 2. Computer software--Development. I. Title.
QA76.73.J38M35 1997
005.13'3--dc20 96-35812
CIP

Printed in the United States of America
10 9 8 7 6 5 4 3 2 1

Who Should Read This Book?

This book is for people who would like to know more about the Java language. If you have an interest in networking, user interfaces, native methods, the Virtual Machine, and the like, this is the book for you.

In general, this book is intended for people with some previous computer experience. If you've ever written a macro in Microsoft Word or Lotus 1-2-3, you should be able to follow most of the material in this book. If you have any doubts, try this simple algorithm:

1. Pick a chapter in the book that sounds interesting to you.
2. Start reading.
3. If you can't understand what you're reading, pick a point halfway between where you are and the beginning of the book and go back to step 2.
4. If you end up back at this paragraph, put this book down and find another.

If you are unfamiliar with object-oriented programming (OOP) concepts, start with Chapters 2 and 3. These chapters should teach you how to set up your Java environment and the basics of OOP in short order, and then you can move on to the specifics of Java language. Then you should be prepared to tackle the rest of the book.

Some of the other chapters include the following:

Chapter 4: Simple Java Applications and the java.lang Package. This chapter focuses on the core of the Java class library. These are the classes you'll end up using on a daily basis.

Chapter 5: Applets. This chapter discusses the specifics of applet programming and introduces graphics and fonts.

Chapter 6: Inside AWT. In this chapter you'll learn about the Abstract Window Toolkit, Layout Managers (including how to write your own), Components and Containers, events, and event handling.

Chapter 7: A Tangled Web: Java Multithreading. This chapter covers threads and multithreading in Java. Special attention is paid to critical sections and thread synchronization.

Chapter 8: The Java I/O Package. This chapter covers Java I/O and the java.lang.io package, including general stream and file I/O.

Chapter 9: Java Networking. This chapter shows how to make your programs "networthy" and how to use network connections and Remote Method Invocation. As a bonus, you will learn how to steal processor cycles from everyone browsing your Web page and use them to factor large numbers.

Chapter 10: Native Methods. First you will learn how to call Native methods from Java and pass data to them. Then you'll learn how to call back from a native method into Java. Finally, we'll look at the deep voodoo of the Virtual Machine.

Chapter 11: Internet Capitalism: Shopping Carts and Databases. In this chapter we examine two commercial applications for Java: virtual shopping and database access

About the Author

Joshua Marketos is a programmer, propagandist, and singer-songwriter from Providence, Rhode Island. He is currently head of research and development for SMT. Recent projects have included the Shemp mailreader, the Shempscape WWW browser, the ShempIt and UnShempIt compression utilities and the Shempcryption encryption standard. When not playing with his avant-garde rock-n-roll band, Schwa, he can usually be found writing songs about United Nations black helicopters and attempting to rule the world from his desktop.

Acknowledgments

I'd like to take this opportunity to thank all the people who made this book possible. Brian "Colonel Panic" Jepson convinced me to write this book in the first place and so shares much of the credit and/or blame. Sean "Dr. Cretog" O'Neill, my attorney and fellow member of the band Schwa, helped to temper some of my abuses of the English language, doctored ailing code, and was of great help in the preparation of many of the tables for this book. Scott "Cool Mafia Nickname" Schoen was the impetus behind the Shempcryption project and was there with the hardware when the inevitable system failures occurred. Wayne "Sixty-Four Megabyte" Alvarez helped by lending me his experimental computer for testing (is that chip still classified?).

The other members of Schwa, namely Motom Boutique, MC Schwa, and Major Hemisphere, helped out by giving me something to keep my mind off Java for a while when my brain was in danger of overloading. The members of SMT also played a big role with their constant nagging and demands for updates on the status of the book, especially Shawn "I am the Walrus" Wallace and Bert "Artistic Expression" Crenca (who I am convinced to this day doesn't even know what Java is). Last, but by no means least, I must thank the folks at Wiley for their patience and for putting up with the corrupted and/or virus-infected files I sent them. It wasn't intentional. Honest.

Joshua Marketos

Providence, Rhode Island

About The Web Site

You can download the source code for any of the applets/applications in this book through our World Wide Web site. The URL is www.wiley.com/compbooks. The Source listings are all contained in one zipped file.This site also contains examples of some of the applets at work, plus some links to various Java resources on the Web. Enjoy!

Contents

Politics and the Java Language

Zen and the Art of Software Maintenance

Java had its start in 1991 when James Gosling and a team of Sun engineers were developing software and operating systems for consumer electronics. They started out using C++, but found it lacking in several critical respects. The Sun team decided to take a leap of faith and create a new language from scratch to support the features they needed. At this point there was no thought of Web pages or applets or taking over the world from their desktops—just a desire to get the job at hand done in the best possible way. From this primordial, nutrient-rich ooze sprang Java, and in the few years since, Java's growth has been nothing short of phenomenal, though, ironically, the consumer electronics market is one of the few areas where Java has not yet caught on.

A recent search of the World Wide Web and Usenet using the Altavista search engine revealed 1,574,406 occurrences of the word *Java.* Still, peer pressure alone is no reason to switch from what you are using now. After all, if all your friends started using Pascal, would you do it too? When I'm asked for one good reason programmers should use Java, I respond, "For the Zen-like feeling of inner calm." Sun describes Java as a "simple, object-oriented,

distributed, interpreted, robust, secure, architecture-neutral, portable, high-performance, multithreaded, and dynamic language," and I can't think of a better description of the sound of one hand clapping.

Simple

The basic philosophy the Java team used when designing the language is KISS. As you may know, KISS stands for "Knights In Satan's Service," and it indicates their commitment to the worship of Our Dark Lord and all his minions. *Just kidding!* KISS stands for "keep it simple, stupid." Some indication of this dedication to simplicity is the quote from Antoine de Saint-Exupery included in the Java White Paper: "You know you've achieved perfection in design, not when you have nothing more to add, but when you have nothing more to take away."

Because of this approach to design, Java is very easy to learn. One must realize that much of the initial difficulty in learning object-oriented languages is actually caused by the basic concepts of object-oriented programming, not the basic syntax of the language itself. Therefore C++ programmers should be able to learn the Java language in short order, and Java programmers shouldn't have too much trouble picking up C++. In fact, Java may be the ideal off ramp for those wishing to learn C++, and there isn't much excuse for not learning both. However, a sudden shift from one environment to the other should be undertaken with caution—though Java has much in common with C++, it *looks* more like C++ than it actually *is*.

Interpreted, Portable, Architecture Neutral

Three more adjectives that describe Java—interpreted, portable, and architecture neutral—boil down to the same thing. Instead of compiling to the native instruction set of any given computer, Java compiles to a set of bytecodes that are meant to be interpreted by a *virtual machine*. This method has two advantages. The more obvious advantage is the fact that your "compiled" Java programs will run on any platform which has an implementation of the Virtual Machine. The second, not-so-obvious, advantage of this scheme is that productivity is increased over that of the traditional "edit-compile-link-load-test-crash-debug" cycle. The bytecode is type-checked and everything-else-checked at compile time, something which is impossible with a true compiled language.

Object Oriented

Java is not a pure object-oriented language. The basic (known as *primitive,* which isn't exactly politically correct) data types are not objects, and that fact has kept away some of the OO purists. One of the favorite pastimes of many language proponents is playing the

more-object-oriented-than-thou game. Who cares? I believe it was cybernetics pioneer F. Gump who said "Object is as Object does." or was it "Life is like a box of Objects." The most important thing is not how fully "object oriented" a language is, but how well the object-oriented features in it work to your advantage. Java seems to be a winner in this case.

Distributed

Java has the ability to treat objects located across the network as if they were local. Using a standard called RMI (remote method invocation), you can make a method call to an object in Outer Mongolia (almost) as easily as you can to one in the box sitting on your desk. Also, classes may be loaded from a remote machine as necessary.

Robust

I can tell you that Java is robust, but there is no guarantee you'll believe me. Java does extensive checking at both compile time and run time to eliminate type mismatches and other potential problems. Most of the features that could get a program into trouble (e.g., *pointers!*) have been locked away out of reach, and the global memory heap is automatically garbage collected to eliminate memory leaks.

High Performance

High performance is obviously a relative term, but for an interpreted language Java is relatively fast. For an architecture-neutral interpreted language that also keeps you from shooting yourself in the foot, it's astoundingly fast. Benchmarks time in at about 1/10 the speed of compiled C. As long as you're not using your code for Patriot missile guidance systems—oh, I forgot, *those didn't work*—and are writing the most common kinds of applications, those that are interactive or do a lot of I/O or network operations, the difference in speed is not very important. And it's faster than Perl.

Dynamic

No, we're not talking about the personality of the language, but rather about the fact that Java is dynamically linked. New classes are loaded only when needed, and that class loading can take place across the network, if necessary. Throw out your old, outdated linkers!

Secure

In these uncertain times, we all need to feel a sense of security, and a malicious attack on one's desktop can send one falling into a nihilistic downward spiral. Therefore, it's no surprise that *security* in network environments is one of Java's most seductive attributes. Much of this cyberdomestic tranquillity comes from some of the features already mentioned, such as the fact that Java has no pointers (hallelujah!), as well as the lack of