

# JAVA™ DEVELOPER'S JOURNAL

The World's Leading Java Resource

February 2002 Volume: 7 Issue: 2

JAVADEVELOPERSJOURNAL.COM

Download  
HP-AS 8.0  
**FREE**



www.hpmiddleware.com/download

From the Editor  
by Alan Williamson pg. 5

J2EE Editorial  
by Ajit Sagar pg. 8

J2SE Editorial  
by Keith Brown pg. 32

J2ME Editorial  
by Jason R. Briggs pg. 74

Industry Commentary  
by Jeff Capone pg. 76

WebSphere vs .NET  
pg. 102

Cubist Threads  
by Blair Wyman pg. 114










RETAILERS PLEASE DISPLAY  
UNTIL APRIL 30, 2002

\$5.99US \$6.99CAN



SYS-CON  
MEDIA



 <b>Architectural Patterns: Building to Scale</b>	<i>Nigel Thomas</i>	<b>10</b>
<i>JMS and JCACHE - an architectural recipe</i>		
 <b>Feature: Coming Out of the JDO Closet</b>	<i>Yaron Telem &amp; Shay Litvak</i>	<b>20</b>
<i>A promising newcomer</i>		
 <b>Java &amp; XML: Use XML Inside Existing Apps</b>	<i>John Goodson</i>	<b>34</b>
<i>...without learning the XML native programming models</i>		
 <b>Feature: The Open-Closed Principle</b>	<i>Kirk Knoernschild</i>	<b>40</b>
<i>Create systems that are easily maintained and flexible</i>		
 <b>Unit Testing: Test First, Code Later</b>	<i>Thomas Hammell &amp; Robert Nettleton</i>	<b>48</b>
<i>A guide to integrating JUnit into the development process</i>		
 <b>Performance Tips: Dynamic Code Generation</b>	<i>Norman Richards</i>	<b>54</b>
<i>Dynamic behavior without a large performance penalty</i>		
 <b>Generative Programming: Under the Hood:</b>	<i>Paul McLachlan</i>	<b>64</b>
<i>java.lang.reflect.Proxy Think more, type less, use a Proxy</i>		
 <b>Acceleration: Boosting the Performance of Java Software</b>	<i>Ron Stein</i>	<b>90</b>
<i>Using Java technology on a portable device</i>		
 <b>Java &amp; JBuilder: Using the JTable</b>	<i>Bob Hendry</i>	<b>98</b>
<i>A step-by-step guide to updating the database</i>		

# **sonic**

[www.sonic.com](http://www.sonic.com)

# zerog

[www.zerog.com](http://www.zerog.com)

**bea**  
[www.bea.com](http://www.bea.com)

## INTERNATIONAL ADVISORY BOARD

- CALVIN ALSTIN (Lead Software Engineer, J2SE Linux Project, Sun Microsystems),
- JAMES DUNCAN DAVIDSON (JavaServlet API/WMP API, Sun Microsystems),
- JASON HUNTER (Senior Technologist, CollabNet), • JON S. STEVENS (Apache Software Foundation),
- RICK ROSS (President, JavaLobby), • BILL ROTH (Group Product Manager, Sun Microsystems), • BILL WILLETT (CEO, Programmer's Paradise)
- BLAIR WYMAN (Chief Software Architect IBM Rochester)

## EDITORIAL

- EDITOR-IN-CHIEF: ALAN WILLIAMSON
- EDITORIAL DIRECTOR: JEREMY GEELAN
- J2EE EDITOR: AJIT SAGAR
- J2ME EDITOR: JASON R. BRIGGS
- J2SE EDITOR: KEITH BROWN
- PRODUCT REVIEW EDITOR: JIM MILBERY
- FOUNDING EDITOR: SEAN RHODY

## PRODUCTION

- VICE PRESIDENT, PRODUCTION AND DESIGN: JIM MORGAN
- ASSOCIATE ART DIRECTOR: LOUIS F. CUFFARI
- EXECUTIVE EDITOR: M'LOU PINKHAM
- MANAGING EDITOR: CHERYL VAN SISE
- EDITOR: NANCY VALENTINE
- ASSOCIATE EDITORS: JAMIE MATUSOV
- GAIL SCHULTZ
- JEAN CASSIDY
- ONLINE EDITOR: LIN GOETZ
- TECHNICAL EDITOR: BAHADIR KARUV, PH.D.

## WRITERS IN THIS ISSUE

- JASON R. BRIGGS, KEITH BROWN, JEFF CAPONE, JOHN GOODSON,
- THOMAS HAMMILL, BOB HENDRY, KIRK KNOERNSCHILD, SHAY LITVAK,
- PAUL MCLACHLAN, ROBERT NETTLETON, NORMAN RICHARDS, RON STEIN,
- YARON TELEM, NIGEL THOMAS, ROB TIFFANY,
- ALAN WILLIAMSON, BLAIR WYMAN

## SUBSCRIPTIONS:

FOR SUBSCRIPTIONS AND REQUESTS FOR BULK ORDERS,  
PLEASE SEND YOUR LETTERS TO SUBSCRIPTION DEPARTMENT

SUBSCRIPTION HOTLINE: [SUBSCRIBE@SYS-CON.COM](mailto:SUBSCRIBE@SYS-CON.COM)

COVER PRICE: \$5.99/ISSUE

DOMESTIC: \$49.99/YR. (12 ISSUES)

CANADA/MEXICO: \$79.99/YR. OVERSEAS: \$99.99/YR.

(U.S. BANKS OR MONEY ORDERS). BACK ISSUES: \$10/EA., INTERNATIONAL \$15/EA.

## EDITORIAL OFFICES:

SYS-CON MEDIA 135 CHESTNUT RIDGE RD., MONTVALE, NJ 07645

TELEPHONE: 201 802-3000 FAX: 201 782-9600

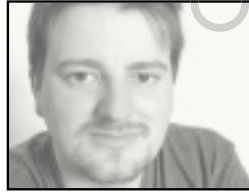
JAVA DEVELOPER'S JOURNAL (ISSN#1087-6944) is published monthly  
(12 times a year) for \$49.99 by SYS-CON Publications, Inc., 135 Chestnut  
Ridge Road, Montvale, NJ 07645. Periodicals postage rates are paid at  
Montvale, NJ 07645 and additional mailing offices. POSTMASTER: Send address  
changes to: JAVA DEVELOPER'S JOURNAL, SYS-CON Publications, Inc.,  
135 Chestnut Ridge Road, Montvale, NJ 07645.

## © COPYRIGHT:

Copyright © 2002 by SYS-CON Publications, Inc. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or by any  
means, electronic or mechanical, including photocopy or any information storage and  
retrieval system, without written permission. For promotional reprints, contact reprint coordi-  
nator Carrie Gebert, [carrieg@sys-con.com](mailto:carrieg@sys-con.com). SYS-CON Publications, Inc., reserves the right  
to revise, republish and authorize its readers to use the articles submitted for publication.

Java and Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc.,  
in the United States and other countries. SYS-CON Publications, Inc., is independent of Sun  
Microsystems, Inc. All brand and product names used on these pages are trade names,  
service marks or trademarks of their respective companies.



ALAN WILLIAMSON EDITOR-IN-CHIEF

# Looks Weird . . . Sounds Great!

A mobile phone (or cell for our American friends) is like a wrist-watch in many respects. You don't change it too often, putting up with its little idiosyncrasies, loving its familiarity; you need something pretty spectacular to lure you away and start the hassle of getting to know another personality.

This month I took the plunge and updated my Nokia to the latest model. I had my previous phone (7110) for a little over a year, and although it was WAP-enabled, it wasn't Java. I appeased my conscience by fooling myself that I was looking at our company intranet, which was at least serving up the WML files from a Java servlet. My new Nokia phone (5110) doubles as an MP3 player. While it has retained the WAP connectivity, there's still no Java. Why?

This phone comes with a full QWERTY keyboard, which makes sending SMS messages a breeze and very fast. The unit is no bigger than my previous phone, which was still a small model. The MP3/radio player on it (64MB) is brilliant and integrates well with the phone functionality, killing the music to accept an incoming call and resuming again after the call ends. So with a QWERTY keyboard and great music why is there still no KVM?

Had there been one, I would have been the first one to write a MIDlet and have it coded up to allow me to make my phone into a cheap version of a BlackBerry, but a much richer one since it could also make voice calls. What is Nokia playing at by leaving out this crucial platform that would allow their phones to become the next handheld revolution?

What do they know that we don't? That's the question I'm beginning to ask. There are

millions of phones in use but very few are Java-enabled. Assuming that people don't upgrade their phones too quickly, and when they do they lack a Java installation, we're talking years here before we can assume that Java is available on every handset. Had the browser market been so slow to pick up on Java as Nokia and others are, we probably wouldn't be reading this magazine today. The availability of a language/platform is one of the key ingredients for success. It's for this reason that Netscape must be saluted and that we have to worry about Microsoft and their upcoming C# on .NET.

Strangely enough, Jason Briggs (our esteemed J2ME editor) has just written an article that looks into this very logistical issue of distributing MIDlets and it struck a chord. Look for it next month.

Speaking of editors, have you read our new J2SE editor's ramblings? Keith Brown wrote his first editorial for us last month and follows up this month with a great piece on the pain of certification. I fear his sentiments are echoed by many who have gone through the same process. Keith is getting up to speed and beginning to shape the J2SE section as per the feedback from you, so keep it coming and let us know what you want to read about.

On a different note, for some strange reason there's been a plethora of jokes this month on everything from Java to the usual digs at Microsoft. However, this one, sent by Gunter Sammet to the Straight Talking mailing list, had me tittering away all day.

*There was once a young man who in his youth professed his desire to become a great writer.*

—continued on page 95

## AUTHOR BIO

Alan Williamson is editor-in-chief of Java Developer's Journal. During the day he holds the post of chief technical officer at n-ary (consulting) Ltd, one of the first companies in the UK to specialize in Java at the server side. Rumor has it he welcomes all suggestions and comments!

[alan@sys-con.com](mailto:alan@sys-con.com)

**together soft**  
[www.togethersoft.com](http://www.togethersoft.com)

A Natural Evolution

While I understand Alan Williamson's lack of excitement concerning Web services, I disagree with his opinion that Web services is a marketing gimmick ("**<Web Services & XML>**" [Vol. 6, issue 11]). We must separate the wheat from the chaff to understand the value of Web services. True, it's about repackaging old applications and technologies, but this is a good thing. It's about repackaging them with new applications and new technologies in ways we never thought possible, with a quickness we never imagined. These old applications and technologies keep the world running and represent billions of dollars invested in IT. Why migrate when you can integrate?

Web services allows for a universal integration of the old with the new. OS390/Cobol, Unix/J2EE, and XP/C# applications all working together in meaningful ways? How is this possible? There has never been a technology so embraced as to make it product-, language-, and platform-agnostic. Rivals such as Microsoft, Sun, Oracle, and IBM all support Web services. The entire industry supports Web services. It will be ubiquitous where CORBA/IOP, J2EE/RMI, and DCOM have failed. It will succeed because it's truly implementation-neutral and runs on the backbone of HTTP and XML.

It's not a revolution. Calling Web services a revolution would make anyone a skeptic. Instead, let's call it a natural evolution. It's middleware for the masses. Let the games begin!



Mel Stockwell  
mel.stockwell@iona.com

A Breath of Fresh Air

Thanks for Ajit Sagar's great editorial "We've Got It All..." (Vol. 6, issue 12). I couldn't believe it when I read the word "WebObjects" in

a **JDJ** article. Why the mainstream industry press consistently ignores the original and best app server in the business, I'll never understand. Your article, however, was a breath of fresh air and hopefully will prove a real eye-opener for frustrated EJB developers looking for a better solution, and cost-cutting managers looking for a better value. Thanks again for helping to get the word out there about a great product. I look forward to possibly more mentions and maybe a deeper analysis of the product in the near future.

Bob Edmonston  
bob.edmonston@shiplogix.com



Interesting editorial about the future of J2EE app servers. We've been looking at migrating our stuff to WebSphere (version 4). While the CMP aspects of the EJB container are appealing, it's still in a pretty primitive state to be useful to us. We're proceeding forward in our investigation - but it'll be a step backward for us to migrate it. The cost to deploy a WebSphere solution is almost \$10,000 per machine (compared with \$700 for WO), the IDE is significantly slower on the same machine, and the runtime environment is much more complex to manage.

Dov Rosenberg  
dov@netcommercecorp.com

Back to Basics

I was happy to read Alan Williamson's comments in his editorial "Back to Work with Ant!" (Vol. 6, issue 12) about moving away from an IDE and going back to basics. I've often felt that I'm the only one who thinks all the wizards, windows, and generated code cause more harm than good.

Many times I see developers using IDEs and not having the foggiest idea what the tool is doing

behind the scenes. Then they try to debug...

Good riddance! Although I prefer a slightly more robust editor such as EMACs.

Tim Thomas  
tim@thomas.net

Just-in-Time Solution

Dr. Java's solution for password authentication for a proxy solved a problem just in time for me (Vol. 6, issue 11). Thanks for the help.

Abhishek Basumallick  
a.mallick@zensar.com





AJIT SAGAR J2EE EDITOR

## Are You Compliant?

In a tough competitive market one of the biggest challenges vendors face is what message to put around their product to distinguish it from their competitor's. The question of which features to focus on is a tough call. The dilemma is paradoxical. With the emphasis on standards, all vendors need to comply with published standards. However, this levels the playing field and leaves very little room for vendors to highlight the functionality that can distinguish them from others. Hence, each vendor needs to provide value-added features that attract clients to their fare.

Lowering prices is one such way to provide the differentiator. However, this may end up being a double-edged sword. If the product is too cheap, or open-source, folks are often put off as they associate quality with price. Regarding standards compliance, one differentiator for vendors is temporal – how fast can you comply?

In the realm of J2EE application servers, standards compliance is given ample attention. After all J2EE is really a set of standards. The reference implementation from Sun is just that – for reference. Application server vendors develop implementations of the standard APIs, tag on their add-on bits, and make the resulting product available to the general J2EE community.

Recently I read a quote saying that Java was dying because vendors add proprietary extensions to the standards. I fail to see the connection. In fact, since vendors provide value-add and there's a need for value-added features, the Java market is growing. On the other hand, the J2EE standards base is also growing, so the need to comply with the latest versions of the standards is crucial for the application server vendors to gain the lead in market share. If a vendor is the first to provide a platform complying with the latest standard, you can bet the corresponding Web site will

declare this with much pomp and show and you'll see qualifiers like "the first and only..." or words to that effect describing the latest product release.

A testimony to the importance attached to being the first in a compliance race is the recent *JDJ Industry News* story on the race for J2EE 1.3 compatibility between the big dogs – BEA and IBM – and a small vendor – Pramati ([www.sys-con.com/java/articlenews.cfm?id=1263](http://www.sys-con.com/java/articlenews.cfm?id=1263)). Pramati is not a market leader in the J2EE application server space, but its recent announcement about being the first vendor to achieve J2EE 1.3 compatibility has made some waves. Soon after Pramati staked a claim on being the first, IBM came out with its own announcement about WebSphere. IBM's announcement leverages their second-place finish as a feather in their hat when compared with BEA, the current market leader in the J2EE space. Remember how BEA had claimed EJB 2.0 compatibility to do the same?

IBM had mistakenly claimed first place. Refuting this has given Pramati more attention (as they are a much smaller player) than the original claim itself. The first spot is more critical for Pramati than for IBM, which already claims 34% of the market share. Pramati Technologies ([www.pramati.com](http://www.pramati.com)) operates out of San Jose, CA, and New York here in the U.S. and has been developing J2EE technology from Hyderabad, India. With Pramati Server 3.0, Pramati hopes to succeed in the mid-market by leveraging the compatibility issue with the low cost of its server as compared to IBM (WebSphere), BEA (WebLogic), and Sun (iPlanet).

Realistically, the issue of who achieved compatibility first doesn't make that much of a difference, as long as the time difference between certifications is not in the scale of several months. If this is so, the application server vendor who is late to the race will have a pretty tough time

### Are You Compliant?

In a tough competitive market one of the biggest challenges vendors face is what message to put around their product to distinguish it from their competitor's.

by Ajit Sagar

8

### Building to Scale

JMS meets JCACHE

by Nigel Thomas

10

### Coming Out of the JDO Closet

The JDO specification is a promising newcomer in the object persistence arena – for J2EE projects large and small

by Yaron Telem and Shay Litvak

20

ajit@sys-con.com

#### AUTHOR BIO

Ajit Sagar is the J2EE editor of JDJ and the founding editor and editor-in-chief of XML-Journal. A lead architect with Innovatem, based in Dallas, he's well versed in Java, Web, and XML technologies.

reestablishing themselves. Compatibility is the bare minimum requirement to stay in the race. However, the quality of the server is what will ultimately decide the success of the product. The good thing about compatibility certification is that it helps separate the men from the boys. ☘



**hp**  
[www.hp.com](http://www.hp.com)

# Building to Scale

## JMS meets JCACHE



WRITTEN BY  
NIGEL THOMAS

**T**here's an old rule in software engineering: "Building to scale requires prior intent." Many applications delivered today fail to address scalability; they get deployed fast and sink faster as the load cripples them.

The advent of J2EE 1.3 goes part way toward providing an environment built to scale. The adoption of JMS- and message-driven beans, as a mandatory addition to J2EE, solves part of the puzzle, but the marriage of JMS with JCACHE (JSR 107) makes life much more interesting.

John Bentley, author of *Programming Pearls*, coined a phrase called the "Ah-Ha!" moment – the moment when you fully understand the dimensions of the problem and, as a consequence, the solution becomes obvious. Those of us who have read the famous Gang of Four book on programming design patterns, *Design Patterns: Elements of Reusable Object-Oriented Software*, will appreciate the elegance and no doubt say, "Of

first commercial implementations of distributed database research and development efforts yielded up much of what we now see as distributed databases complete with replication. These distributed databases reflected the geographic dispersal and management of the information base that defined an enterprise. The techniques employed were synchronous and utilized a simple request/response paradigm for data distribution between database management systems as well as between clients and database servers.

### Real-Time Computing

Meanwhile the real-time computing community was also looking at more asynchronous forms of communication.

### Message-Oriented Middleware

In the 10 years that preceded the birth of Java, a lot of research and products came onto the market to define a new space called Message-Oriented Middleware (MOM) for communicating between processes and applications. The early adopters of this new technology were financial service companies such as Goldman Sachs, Lehman Brothers, and Nomura. In fact, many banks were either trying to co-develop a messaging solution or build it in-house.

Why was MOM so important to the major financial institutions? It enabled them to decouple their systems and provide a flexible solution that reflected their working practices. Furthermore, it

“Understanding a problem is all about understanding its form or shape. When we fully appreciate the shape of a problem we can find a solution that truly fits”

course, I used that pattern on my last project.” The marriage of JMS and JCACHE is an architectural recipe for distributed systems and gives rise to further patterns as we understand the dimensions of the problems we're trying to solve.

In this article we look at how these technologies can be used to provide scalable architectures for delivering J2EE-compliant, Web-based applications today and support what we need to deliver tomorrow by exposing some architectural patterns based on JMS and JCACHE.

### A History of Distributed Systems

The history of distributed systems development has its genesis in the days of database management systems. These first systems were centralized monolithic beasts and, soon after, the

techniques used in the late 1970s and early '80s were largely event-driven and used communication mechanisms for message exchange in multithreaded and multiprocessor environments. Indeed, much of what was done in those days is reflected in the black art that underpins the building of device drivers.

The building of multitier applications and the rise of the Internet has focused a lot of attention on the techniques for building distributed systems. When Java was born it had remote method invocation built into the language. It supported sockets to enable interprocess and intraprocess communication. While the device drivers for socket handling were asynchronous, the standard mechanism for implementing multitier applications remained rooted in a request/response synchronous paradigm.

enabled them to achieve higher trading throughput by managing business transactions as event flows rather than as database transactions. The fundamental shift to business transactions obviated the need to wait for the database to process the entire life cycle of a trade, and allowed it to be decomposed into a number of database transactions that reflected the natural flow involved in recording, accounting, and settling trades.

### Web Services

Web services are the latest and greatest Internet craze. They support the focus for much of the J2EE 1.3 development (with the JAX pack) and will have a huge influence in what is in J2EE 1.4. Web services are service-centric, not document-centric the way the Internet

# **compuware**

[www.compuware.com](http://www.compuware.com)

is used today. Components can be defined as a Web service and so promote reuse. Service definition is through the Web Service Description Language (WSDL) and registration through the UDDI (the repository). Communication is encoded using the Simple Object Access Protocol (SOAP), which is an XML synchronous remote procedure call.

### Architectural Patterns

Where does all this history leave us? And where does JMS and JCACHE take us? Well, we can connect applications pretty effectively. We can even reuse legacy-messaging investments through a more open JMS framework approach and so provide standards-based, multi-plug asynchronous connectivity. On the other hand, the development of SOAP with JMS gives us a standard synchronous functional protocol over an asynchronous communication mechanism in a Web services context. Now JCACHE provides us with a set of interfaces that, coupled with JMS, provide the bedrock for elaborating standard architectural patterns that are geared to alleviating the problem of server bottlenecks.

The technologies we described thus far fall into two camps: those that are based on a (request/response) synchronous paradigm and those that fall into a (publish/subscribe) asynchronous paradigm. Messaging is generally asynchronous. It has to be to promote fire-and-forget messaging and so support a decoupled solution. But applications are generally built using synchronous mechanisms. The challenge is how to combine these paradigms into an architecture that engenders scalability. To do this we need to understand the problems. In part they're a consequence of history, which is why history is important.

In the next section we examine the problems and present some architectural design patterns that are the basis for building scalable, distributed systems.

#### The Role of MOM and JMS

The Java Message Service API was born in late 1997, and the first commercial implementation came out in the spring of 1998 from SpiritSoft. Since then a market has grown up around the standard as Fiorano and Progress (now Sonic Software) entered the fray. IBM offered a JMS interface to MQSeries shortly after, and finally Tibco Software joined the club late in 2001.

The adoption of JMS as the first vendor-independent MOM standard is a testament to the power of asynchronous

messaging as a fundamental technology for delivering distributed solutions. The move toward message-driven beans and the mandatory status of JMS in J2EE 1.3 is a huge step forward. The ability of many JMS products to offer a scalable load-balanced solution to the distribution, marshaling, and management of requests to an application server supports this. However, it's only the beginning. Life gets much more interesting when you start to apply MOM technology through a JMS standard to caching and so start to understand the basic architectural patterns that it promotes.

#### What Does JMS Do for Us?

What is JMS and what does it do for us? (For more detailed information check out the following two books: *Java Message Service* by Richard Monson-Haefel and David A. Chappell, and *Professional JMS Programming* by Paul Giotta, et al.) The Java Message Service API is an application programming interface. It's defined as a set of interfaces for producing and consuming messages based on publish and subscribe as well as point-to-point queuing models. The JMS specification enables subscribers or receivers to specify a quality of service for messages so they can be reliably delivered or guaranteed to be delivered. It's similar to guaranteed versus registered delivery by the postal service. The JMS specification also describes some semantics that JMS vendors must conform to, such as how a message is received. Thus a queue allows only one receiver to receive a particular message (i.e., it's delivered once and only once), whereas many subscribers to a topic might receive the same message.

#### JCACHE and the Role of Caching

Although JCACHE (JSR107) is still making its way through the Java Community Process, several vendors have announced or will be announcing caching products based on it. What JCACHE will provide is a standard set of APIs and some semantics that are the basis for most caching behavior. According to its functional requirements, JCACHE "allows applications to share objects across requests, across users, and coordinates the life cycle of the objects across processes."

JCACHE provides a fairly rich set of APIs that let you exercise control over cache loading, cache eviction, and cache validation. It deals with basic caching patterns in which caches can be fed by other caches. A consequence of this is that it allows implementers to

take advantage of distribution technology, such as that offered by JMS, as well as any other distribution technology deemed appropriate.

According to JCACHE, a cache is "specific to each process." This seemingly simple statement has profound implications and is one of the reasons why the proposed solution is flexible. Each process can implement its own mechanisms for cache loading and replace or share the necessary functionality as required. When an object is invalidated or updated in one process, the name and associated information can be broadcast to all other instances of the cache; this is where JMS comes to the fore, by acting as the standard distribution interface for a cache. This in turn allows the entire system of processes to stay synchronized without the overhead of centralized control.

The long and the short of it is, JCACHE is a smart approach for read-only data.

#### Shape

Understanding a problem is all about understanding its form or shape. When we fully appreciate the shape of a problem we can find a solution that truly fits. It's the difference between effecting a cure and relieving symptoms. All of us have been pressed for time on some software project; a bug has been holding us up for days and we decide to change this or that based on intuition and the problem goes away. We may have fixed the problem, but since we don't understand the shape we can't be sure. On the other hand, a bug holds us up for days and suddenly, perhaps because we weren't looking too hard, we have further insight and understand the shape of the problem. It's an "Ah-Ha" moment; when we truly understand the nature of a problem so the solution hits us in the face.

The problem with the synchronous and asynchronous paradigms is that enterprises are not totally synchronous or asynchronous. They reflect both paradigms in specific ways that support their business models. The shape is there and the architecture needs to reflect it. In this way we can build systems that reflect and support the business rather than the business being a reflection of the computer systems that support it. A synchronous solution would necessitate all components of an enterprise to participate in some form of distributed transaction – a legacy from history, encompassed by the old mantra that "all solutions start with a database," whereas solutions are bounded by a problem.

# **infragistics**

[www.infragistics.com](http://www.infragistics.com)

**Server Bottlenecks**

Request/response systems are by their very nature pull-based and passive. The applications pull data from the server. Many of these applications manipulate the same logical data numerous times within the same period and across many transactions. As the client population increases, so does the number of requests to the application or DBMS server. The server becomes the bottleneck, swamped with too many spurious requests. Application server, DBMS, and hardware vendors suggest using replication to alleviate the problem and thus reduce the client-to-server ratio. This just relieves the symptoms and does nothing about effecting a cure. As the enterprise grows, so does the need for more replication and more servers, and so the costs escalate.

Web services doesn't fare any better because it's synchronous. It does help deliver systems, getting you there fast, because it offers standard protocols and standard interfaces, but it does nothing to alleviate the inevitable problems of server bottlenecks in large-scale Web services within and between enterprises.

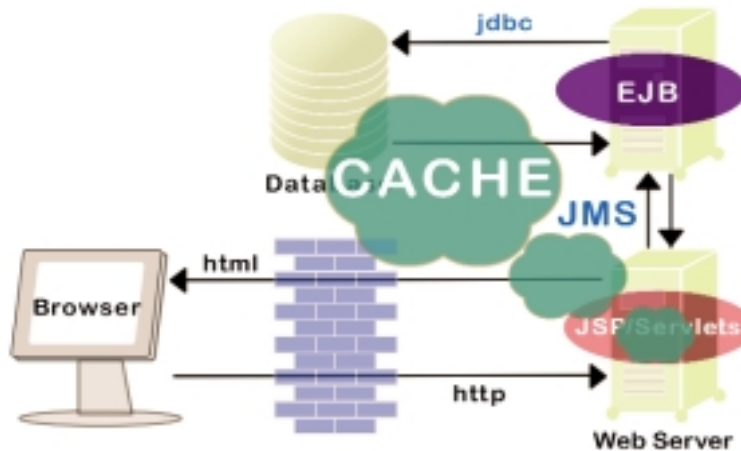


FIGURE 2 Read-only cache

a price change we might be willing to accept the last price published (see Figure 1). In this way each functional unit of the business is responsible for its own scalability issues. The use of JMS enables them to proceed at the rate they need to. The only functional unit that dictates overall performance is the order entry system, which is now free to enter orders without worrying about the other functional units of the enterprise. The flow of messages (or events) can then be controlled as a system-to-system workflow and so provide a better controlling and monitoring environment.

**Simple Caching**

The next architectural nuance is to recognize that many systems spend a high percentage of time reading and then having flurries of activity as they transact and data is changed, added, or deleted. During the mid-1990s, when financial services took up the challenge of decoupled architectures, many also used MOM to provide a distribution mechanism for data. It was a natural step to try to cache the data nearest the application that needed it, and further

reduce the server bottlenecks.

This next pattern simply recognizes that a typical layer between a consumer of messages from a JMS bus and the bus itself is a read-only cache (see Figure 2). The read-only nature of a cache is important here because the cache is decoupled from the underlying data source. Updating the objects doesn't change the source; updatability is a more advanced pattern. What this pattern requires is a container that understands what we want, how to get it, and how to propagate the changes that it receives. This is the basis for most caching behavior.

Once we take the view that the applications can have a cache of information from the application server or DBMS, and that the cache can be up-to-date or active, then the application no longer needs to pull data that it has already requested. Applications in an active architecture pull once for any given object (or objects), and the server simply pushes thereafter. This seemingly simple change has a major impact on performance and scalability and a major impact on the design of the applica-



FIGURE 1 System-to-system workflow

**Transactional Islands**

The first pattern is a reflection of how many organizations can be broken down into logical units. In banking this was always based on splitting the enterprise into front-, middle-, and back-office functions. Each functional unit owns its own transactional space. Each transaction space is then connected by a JMS bus. Depending on the semantics of the messages published on the bus, different qualities of service (QoS) may be used. Thus for an order we would want to ensure it always gets delivered, but for

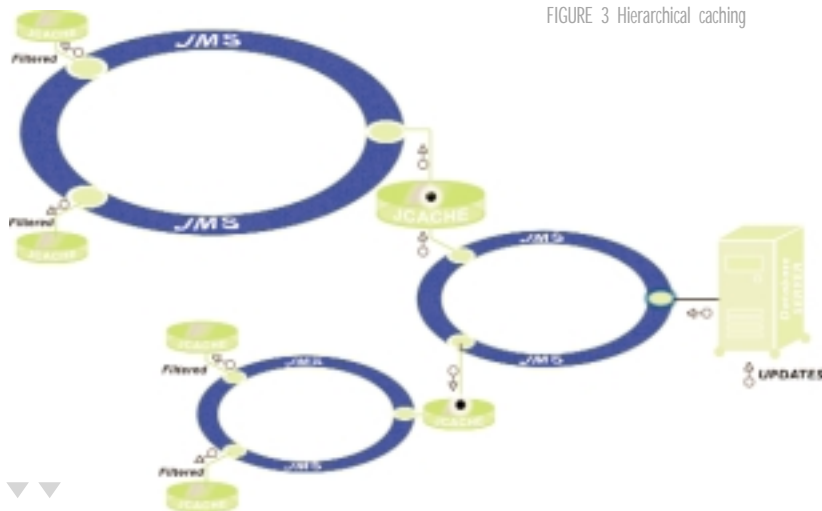


FIGURE 3 Hierarchical caching

**silverstream**  
[www.silverstream.com](http://www.silverstream.com)

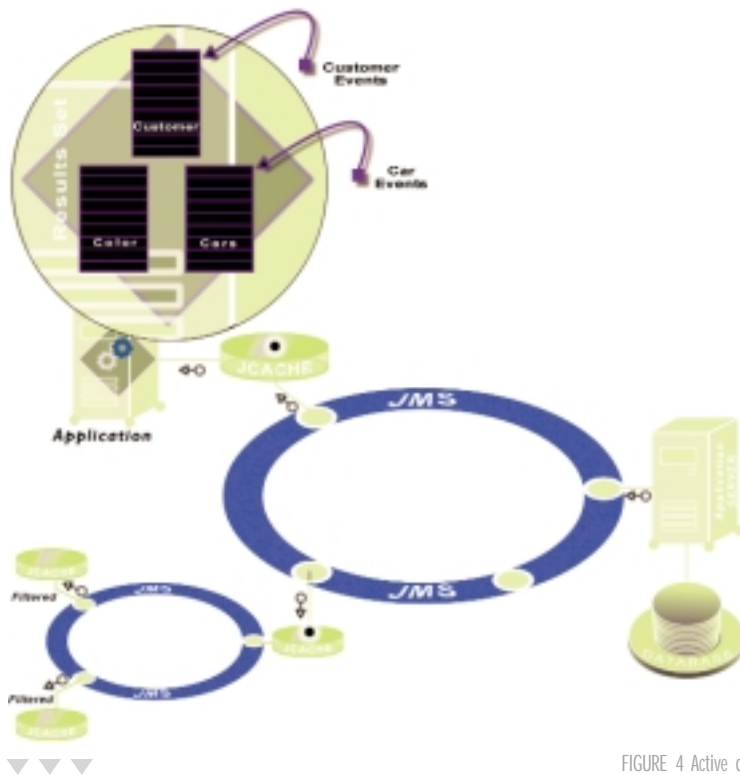


FIGURE 4 Active queries

tions. Applications now need to become event-driven and so, themselves, become active.

**Hierarchical Caching**

A more complex form of caching can be derived from Figure 2. If caches can be connected in a hierarchy with each cache subscribing to a JMS bus (see Figure 3), we can use the natural hierarchical shape of an enterprise and some

counterintuitive logic to model a hierarchical caching solution across an *n*-tier architecture.

In an *n*-tier architecture, the nearer you are to the data source, the finer the granularity you need to identify the data you want. The further away you are, the more localized the data becomes. Consider this: as all the requests for data converge onto a data source, the data source needs to identify what each

requestor needs. But further out toward the edge, each requestor already knows what it needs, be it through object identity or predicate definition. Thus the counterintuitive principle is to have fine-grain subscription closer to the data source and coarse-grain subscription the further away you go from the source. The fact that JMS-based solutions offer topic hierarchies lends itself to this form of traffic shaping and complementary hierarchical caching.

**Active Queries**

The more the active cache is decoupled from the server, the more scalable the solution becomes. Some systems like HOODINI (a distributed investment banking architecture at Nomura Research Institute) implemented a hybrid (push and pull) approach. In HOODINI, applications registered interest through predicates (active queries) and so enabled the cache to be filled based on a very flexible declarative approach. Although the approach in this case was a hybrid, it has considerable merit as it not only used MOM to distribute change, but also provided a flexible way of describing what you want in your cache through the use of predicates.

Active queries and the use of predicates to define what we want in a cache changes the way in which we build applications. Subscription to a cache and the cache's subscription to a data source are based on predicates. Notification of change is based on specific objects or entities and the results set that they're in or have moved to (see Figure 4).

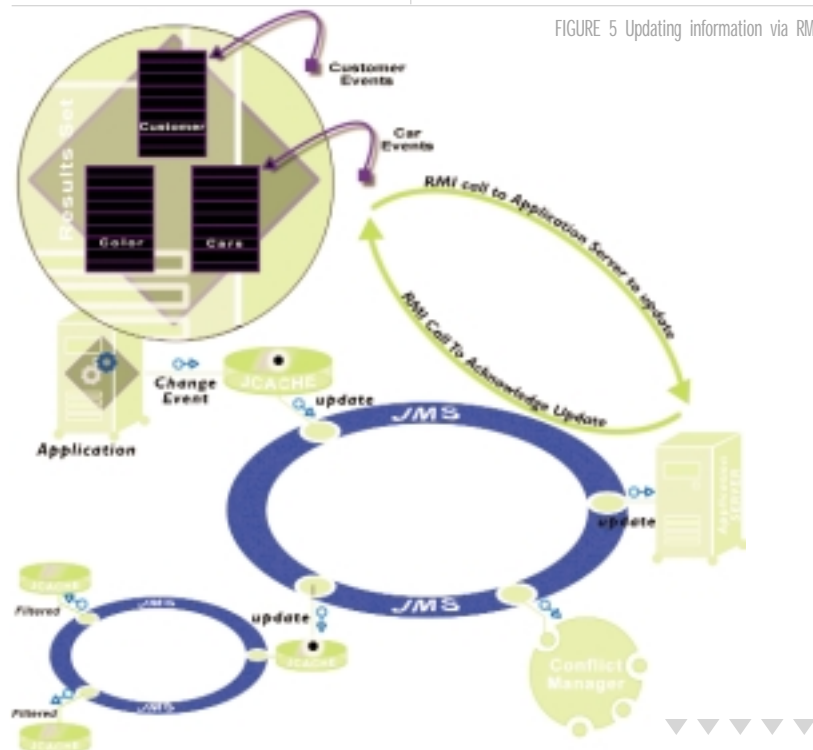


FIGURE 5 Updating information via RMI

**Updatable Caches**

Decoupled caches don't lend themselves to conventional methods of updating the information. The fact that they're decoupled means that the transaction consistency of the cache is sacrificed. Conflicts that arise from inconsistency need to be dealt with as exceptions in the business flow that underpins an enterprise. This is what many financial institutions do anyway and what transactional islands recognize. What we can do is provide patterns that ensure that the window of opportunity in which these problems may arise is as small as possible. This way we can create an architecture in which exceptions are exceptional and most of the investment in software deals with the bread and butter. So how do we close this window? There are two basic mechanisms.

The first uses the concept of a proxy to provide a synchronous update mech-



# sitraka

[www.sitraka.com](http://www.sitraka.com)

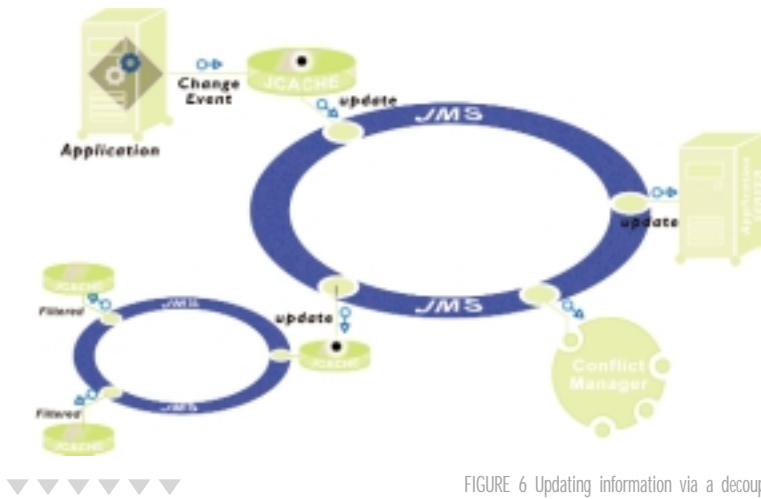


FIGURE 6 Updating information via a decoupled cache

anism. If the business object or entity is a proxy within the cache, and that proxy has all it needs to update the information source, then the proxy can update the information source directly (in standard RMI-type fashion). When the synchronous update call returns, the proxy can publish the update through the cache to other interested parties (see Figure 5).

This pattern is fine for low frequency updates. The synchronous calls to the information source guarantee that all updates are synchronized and so are transactionally safe; however, the price to be paid is the latency of the synchronous call and the attendant server bottleneck. While the read frequency remains high and the update frequency low, this pattern has considerable merit. Indeed this was how the HOODINI cache worked.

The second uses a totally decoupled cache for read and update. When caches are loaded, the information source is no longer the source but is simply treated as a subscriber to change. Changes occur in the network by publishing the changes onto the network, but the information source uses a durable subscription to ensure that it updates the information source (see Figure 6). This carries with it a need to perform conflict resolution when updates to the same object overlap; this can be easily detected by carrying the pre- and postimages of the object, but the conflict resolution is application-defined.

This pattern works for both higher read and high update frequencies, but it does incur the cost of conflict resolution. As long as the number of conflicts is low, then the conflicts can be costed as part of the overall exception handling that systems generally have to perform. If the update profile is segmented so

that different parts of an organization update different data, then the pattern is very effective and increasing overall throughput in the face of change.

### Raising the Debate

How does this fit into the real world? All the patterns mentioned, from transaction islands to decoupled updatable caches, have relevance to today's Web applications as well as to the Web services of tomorrow. The inability of Web servers to cache information effectively suggests that caching based on a JMS distribution model is highly applicable. The same would be true of a SOAP-based Web service as well as for any internal application in which reading is a large part of what an application is used for.

What JMS has done is raise the debate. We are now able to identify patterns, expose further APIs, and develop the related JMS tools market. It moved the debate beyond JMS in the same way that the introduction of SQL gave rise to database tools in the 1980s.

### The Network Is the Database

The adoption of JMS as part of J2EE 1.3 and the widespread use of JMS as a key component in building distributed systems is starting to give rise to a "beyond JMS" debate. What do we need on top of JMS to more effectively deliver systems? What we have argued so far is that JMS and JCache go together and that decomposing a business into transactional islands is a natural fit with JMS. We've also pointed out that business transactions are a natural consequence.

Scott McNealy said, many JavaOnes ago, that the network is the computer, and in a sense it's true. We can go further and contend that the network is the database. Certainly when we navigate the Web we're not aware of any single

data source. We simply state our criteria and hope to get a match that has meaning. The Web provides a static view of data; caching and JMS- and predicate-based subscriptions enable us to apply the same mechanism for Web navigation to changing data. Not only does the network become the database, but it also becomes an agent for managing change.

### In Search of Greater Flexibility

The marriage of JMS and JCache and the use of topics and selectors to provide an open standards-based solution to a common architectural problem offer an "Ah-Ha" moment. It allows us to abstract out some of the architectural patterns that support the construction of large-scale distributed systems. For example, the JMS selectors are what enables the messaging and caching to deliver what we need based on a predicate. It's simply the ability to distribute change events, apply a condition to the change, decide what to change in the cache, and then inform an application of the change to the cache. It's a case of managing events, conditions, and then actions and applying it to JMS and JCache. Flexibility, the provision of runtime change to such approaches, is what we all seek.

We can encapsulate this flexibility by using event-condition-action rules to externalize those parts of the system that need to be on the surface so they can be changed while the system runs. ECA rules enable this to happen. The ECA rules as espoused by RuleML, also known as reactive rules, can be generative (e.g., compiled into Java) and can be small enough to coexist with a cache. Put them together and you have a flexible dynamic mechanism for active caching based on predicates.

The openness of JCache with its policy-driven architecture for cache loading, validation, and replacement can be made totally dynamic by using ECA rules to manage the policy externally to the cache. JMS offers a mechanism to update the cache, providing the data as events to which the ECA rules are applied, and also provides the distribution mechanism for rules to be updated remotely so that caching and rules can be changed wherever they are. In this way it's possible to build robust, scalable, and highly dynamic distributed architectures that can reflect the needs of an enterprise on day one and continue to reflect their needs in the future. ☉

nigel.thomas@spirit-soft.com

#### AUTHOR BIO

Nigel is director of product management at SpiritSoft with over 20 years' experience in the industry, specializing in distributed systems architecture and audit.

# **datadirect**

[www.datadirect.com](http://www.datadirect.com)



J2ME



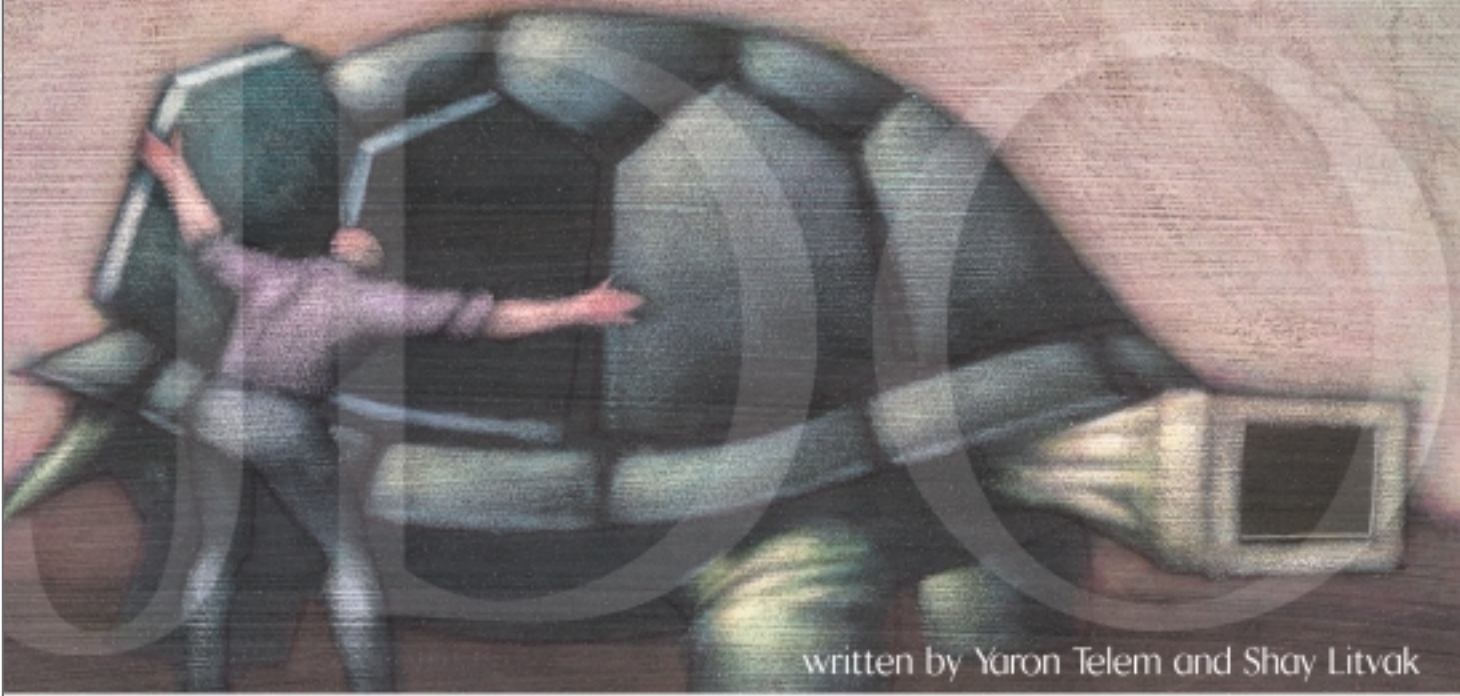
J2SE



J2EE



Home



written by Yaron Telem and Shay Litvak

# Coming Out of the JDO Closet

The JDO specification is a promising newcomer in the object persistency arena

As part of building the infrastructures for a large J2EE project, we've spent the last few months designing and implementing a JDO-based O/R persistency framework. This framework provides our business logic programmers with the following features: an interface-based abstract view of the data-layer with full OO semantics, zero-need knowledge of the object-to-database mapping details, "delta" support, and more. This article presents the persistency framework that we've built on top of JDO, and offers a commentary on the current advantages and shortcomings based on our experience with the JDO specification.

## Choosing an Object Persistency Layer

Choosing the right object persistency layer is a major milestone in almost every J2EE project. Given the state of today's market, you should follow the "buy, don't reinvent" rule (assuming your object persistency demands haven't reached the "you must be dreaming" line). But though today's component server-side frameworks surely make the architect/developer's life simpler, there are still some issues to consider when designing your data layer, namely performance, sound API toward the business logic, and standardization. Limiting ourselves to the object-relational mapping market per se, the choice is between:

- Using a proprietary yet powerful and proven tool from one of the veteran O/R mapping vendors such as TopLink ([www.webgain.com](http://www.webgain.com)), CocoBase ([www.thoughtinc.com](http://www.thoughtinc.com)), or Castor (<http://castor.exolab.org>)
- Relying on one of the newer yet standard object persistence Java specifications: EJB entity beans (ThoughtInc., WebGain, and others are offering entity beans-based solutions as well) or Java Data Objects (JDO)

While all the above vendors offer (more or less) the features listed in Table 1, since we're especially zealous about standardization and vendor independence, we decided on JDO, a strategic decision. Left with choosing between entity beans and Java Data Objects (it was about a year ago that we made this decision), we went again for the second choice. Entity beans seemed rather too bulky and heavyweight for our purposes, obscuring the simple Java model while providing (unnecessarily in our case) factory-wrapped network interfaces and limited inheritance capabilities. We felt that JDO provided a simpler, cleaner API with pure OO syntax. Still another advantage of JDO lies in its relative lightness. (Note that by saying that we're certainly not trying to slight EJB entity beans. Indeed, the initial doubts about their true usefulness seem to have died out, especially after the improvements introduced in the EJB 2.0 spec.)

# **altoweb**

[www.altoweb.com](http://www.altoweb.com)

Data layer exporting OO semantics and API  
 Transparent mapping of the objects to a relational datastore, supporting varied object relationships (at least: 1-1, 1-M, M-1, M-M, embedding)  
 Life cycle management and state interrogation  
 Some sort of an object-query language  
 Seamless integration with the J2EE transaction demarcation

TABLE 1 Requirements of an O/R persistency layer

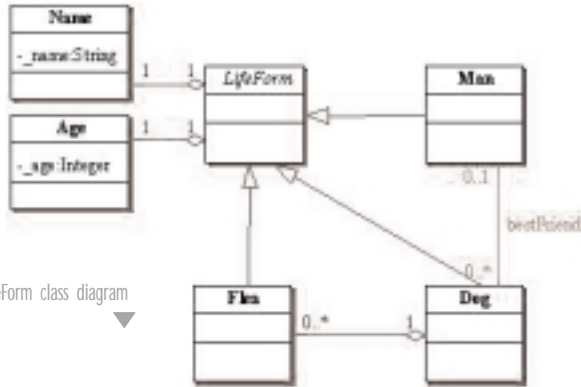


FIGURE 1 LifeForm class diagram

### Short Overview of JDO

In the remainder of this article we'll share the experience gained while building an O/R persistency framework on top of JDO, designed and implemented in a large organization. Let's start by briefly summarizing the main features of JDO.

The JDO specification provides a flexible method for handling persistent objects. It originates from JSR-000012, approved in July 1999 – those not yet familiar with the spec should consult the “Resources” section of this article for the appropriate URL – and was designed to integrate with existing frameworks (most notably J2EE), offering more complete transactional facilities. Indeed, session beans can directly manipulate JDO persistent instances, while entity beans can delegate business logic and persistence management to JDO classes. In our project we took the first approach.

Working with JDO, classes that can be made persistent implement the PersistenceCapable interface, but the class author need not explicitly declare this in the code. All a developer needs to do is to provide a companion XML file, which contains metadata about the class (e.g., the names of the fields that are to be made

persistent). This XML metadata is used by a “class enhancer” to generate a persistence-capable copy of the original class. (JDO ensures binary compatibility between persistence-capable classes, even if generated by different vendors’ enhancers.)

As a result, all the details of the class’s actual persistence management happen behind the scenes. Each JDO instance of a persistent class has its own unique identity in the datastore and hence can be shared between different applications concurrently. When a reference is followed from one persistent class to another, the JDO implementation transparently instantiates it as necessary into the JVM. When fields of a persistent object are accessed or modified, the JDO implementation transparently marks them for change in the datastore. Almost all user-defined classes can be made persistent. While this excludes most system classes (like those in java.io or java.net), some are handled specially by a JDO implementation. For instance, the java.util.Collection interface is supported, as are immutable classes like Integer, Float, String, and Date – in addition to arrays.

The PersistenceManager is the primary interface for JDO-aware applications. It enables users to store and fetch persistent objects from the datastore and allows users to perform synchronized transactions and queries on those objects. JDO uses a neutral query syntax similar to OQL (Object Query Language).

### Pros and Cons of JDO

The JDO specification has honestly won its territory, being a compact, smart, and truly pure object-oriented persistency specification. However, being a newcomer also means being young, suffering from childhood fallacies, and, sadly enough, having older enemies.

JDO touts as an advantage its back-end and datastore neutrality. This is indeed a virtue of the specification. But on the other hand it entirely ignores various important O/R issues such as object locking paradigm and O/R mapping instructions (e.g., schema structure, referential integrity, constraints, and so on). That is, JDO can be categorized as a general object persistence framework, with no special interest in being “baptized” as a full-blown O/R persistence specification. Furnishing no O/R mapping guidelines (let alone standards) and leaving this important role to the different product vendors makes the application developer’s life somewhat confusing. For instance, Kodo ([www.techtrade.com](http://www.techtrade.com)), ObjectFusion ([www.prismtech.com](http://www.prismtech.com)), and IntelliBO ([www.signsoft.com](http://www.signsoft.com)) don't use database-referential constraints. But other vendors may choose to do so. And while IntelliBO lets you define code mappings (translation of predefined values into more compact database representations), it requires you to use proprietary XML tags to do it.

Aside from its current “O/R uncertainties,” there's still some uncertainty surrounding JDO. Sun, by not making a very good job of telling us when to use EJB entity beans and when to use Java Data Objects, has added to the overall confusion. Furthermore, the current JDO vendors tend to be “niche vendors” in the Java industry. A major step toward the acceptance of JDO would happen when the “big guns” like IBM, Oracle, and BEA begin to follow suit and join them – something that's not yet happened. Finally, JDO has also managed to win itself some sour enemies. The infamous duel on TheServerSide.com between JDO specification lead Craig Russel and Thought Inc. CTO Ward Mullins is just one example (see “Resources”).

Balancing the pros and cons of JDO, we believe that in spite of its current shortcomings JDO will mature into a pervasive first-class object persistency solution. An up-to-date portrait of the JDO market can be found on the JDO official site (see “Resources”).

### A Simple Approach to Working with JDO

As can be seen in our UML class diagram (see Figure 1), the superclass of the entities in our example is LifeForm. Man,

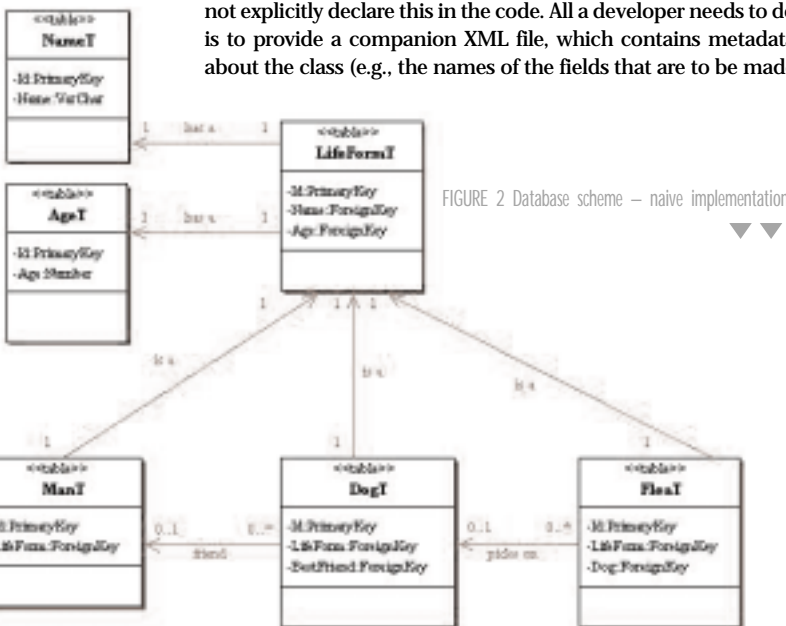


FIGURE 2 Database scheme – naive implementation

# object design

[www.objectdesign.com](http://www.objectdesign.com)

Dog, and Flea are all LifeForms, where Dog also references a collection of Fleas and holds a reference to his best friend, Man. Name and Age are aggregated types in a LifeForm where an Age is always positive and a Name consists entirely of English letters. Note that Name and Age are “primitive wrappers” (of a String and Integer, respectively) supplying us with better abstraction, type safety, and basic validation in the type itself (validated during construction).

The Java code pertaining to the UML specification is presented in Listing 1. Examining the syntax of class Age, a question arises: Is the use of an Integer necessary? Can't we simply use an int? Using the latter would, curiously enough, make it impossible to create an ageless life form, since there's no Java null value for int, thus no datastore NULL value for age. Of course, you can allocate a prespecified int value representing null (e.g., -1) but this would make writing your queries a tad peculiar.

Writing the Java code, our next step is to run the vendor's JDO enhancer tool on the classes. Using the XML metadata file supplied in Listing 2, the enhancer will make these classes PersistenceCapable by applying direct bytecode manipulation. Some enhancers will automatically generate DD SQL statements (or directly create the database schema given a JDBC connection) as part of the enhancing process, while other vendors will politely ask you to run their schema-building tool. Notwithstanding, some vendors let you capture a preexisting schema and declare the O/R mappings yourself.

Robust OO models should follow the real world. Thus, when our dog passes away, its fleas should gracefully pass along with it. The dog's best friend, however, won't (we hope). In JDO terms (see Listing 2) Dog and Man are First Class Objects (FCOs) – i.e., they possess an independent life cycle. On the other hand, Fleas are embedded in Dog as Second Class Objects (SCOs). SCOs have no JDO identity of their own

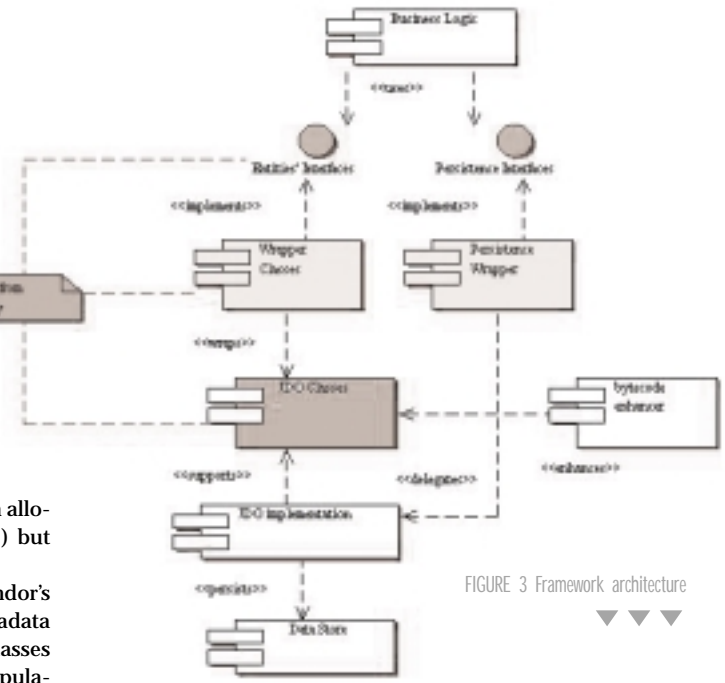


FIGURE 3 Framework architecture

and are stored/deleted from the datastore as part of their owning FCO. To complete the picture, note that Age and Name are embedded as SCOs in LifeForm.

Finally, some business logic code using our data objects can be found in Listing 3. Observe that in deleteDog(Dog) it's enough to explicitly delete the Dog object. Its fleas are implicitly deleted from the datastore along with its Name and Age objects.

### Performance Analysis (Round 1)

Albeit simple, the naive implementation results in poor runtime performance. The database tables generated by a typical schema tool are depicted in Figure 2. The exact table and field names are, not surprisingly, vendor-proprietary as are the referential constraints imposed on the tables. Nevertheless, they would follow the general structure presented here. Looking at Figure 2 we can see that fetching a Dog object results in a minimum of four fetches and three joins (assuming no fleas, no friend, but proper identification). Fetching a LifeForm translates to three fetches and two joins. This is highly inefficient.

You're probably curious why Name and Age aren't represented as columns in LifeFormT, even though they've been declared as “embedded” SCOs. Well, in order to be consistent with the JDO specification, an O/R vendor must make the life cycle of Name and Age dependent on their owner, but there are no restrictions regarding the actual O/R mapping. All the vendors we've tested have failed to exploit the “embed” hint into a smart datastore embedding.

It should be clear that the issue that causes a problem isn't the creation of the NameT and AgeT tables. Since a vendor's knowledge of the application semantics is limited to that defined in the metadata XML file, it would rightfully choose to generate the NameT and AgeT tables. Indeed, Name and Age could be used as an FCO or as elements of Collection in another part of the application. It's the lack of O/R embedding capabilities in cases analogous to LifeFormT that causes the problem.

### Introducing Persistency Framework Based on JDO

Looking back at the naive approach above, you can spot a three-layered persistency architecture consisting of the JDO enhanced classes, the JDO implementation, and the datastore

Our JDO-based persistency framework introduces two additional layers, namely the interface layer and the wrapper layer, thereby attaining the following added-value goals:

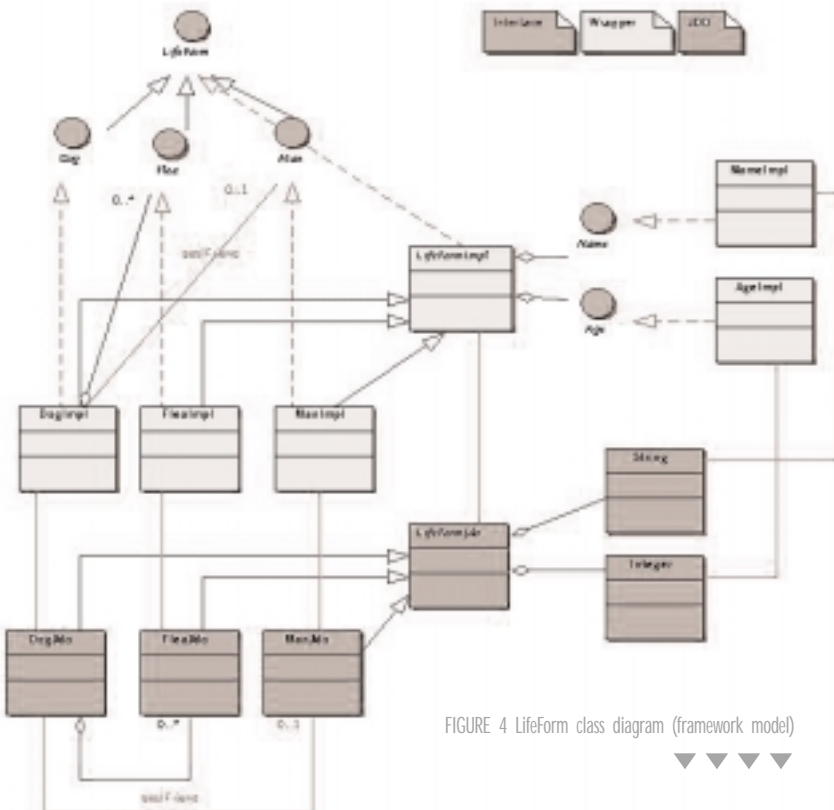


FIGURE 4 LifeForm class diagram (framework model)



# borland

[www.borland.com](http://www.borland.com)

- A: Interface-based abstract view of the data layer
- B: Tuned O/R mapping providing superior performance
- C: "Delta" support and Delta-driven semantics (described below)
- D: Change tracking during the transaction

Figure 3 shows the architecture of our framework. A vertical view of the framework reveals two major components: data entities (generated from the repository) and persistency management (an adapter, delegating the application persistency instructions to the concrete JDO implementation – e.g., persistence by reachability, JDOQL query engine, integration with EJB container, and so on). The horizontal view unveils the framework's layers: interface layer, wrapper layer, JDO layer, JDO implementation, and datastore.

We won't go any further into the persistency management component, since it's mainly just an adapter. Readers eager to learn about persistency features are wholeheartedly directed to the JDO specification.

Figure 4 presents the life form example, refactored to fit our framework. Relations connecting different layers are, as you can see, framework driven, while relations within a layer are OO-model driven. In fact, each layer encapsulates a "replica" of the life form model, using its own abstraction level to do so. Note that whereas Dog, Flea, and Man are explicitly represented throughout the layers, Name and Age are missing from the JDO layer. The reason for this will be made clear later.

#### Interface Layer

It's hard to overstate the importance of using a metadata repository within any large J2EE project. Our project uses one ("home made") in which we define our data entities, their fields, properties, and much more. The metadata repository is the source for our active code-generation tools, which generate our entities' interfaces and implementations and also much of the framework's generic code.

The interface layer (see Listing 4) provides a façade toward the business logic, allowing it to work in terms of abstract interfaces (Goal A). The clear separation between interfaces and implementing classes is not just a good programming practice, but also allows for a more loosely coupled compilation model. Moreover, as our data entities can implement multiple inter-

faces, we're able to provide different business logic modules with personalized local views of the data and enable the creation of generic mechanisms/algorithms operating on interfaces. For instance, one could declare Dog and Man to implement a Trainable interface. Doing that, it's possible to write a generic algorithm working in terms of trainable life forms.

#### Wrapper Layer

The wrapper layer is the middleware between the abstract interfaces presented to the business logic and the persisted objects actually stored in the database. It provides the concrete implementation of the interfaces defined in the interface layer through a symbiotic relationship with the JDO-layer classes (the enhanced classes), which act as the actual value holders. The wrapper layer's chief responsibilities focus on attaining the remaining goals, namely:

- Concrete implementation of the interface layer, employing smart object embedding (Goal B)
- Delta support (Goal C)
- Change tracking during the transaction (Goal D)

Wrapper objects make use of OO mechanisms such as reflection, dynamic invocation, dynamic proxy, lazy wrappings, and more. Their concrete code, being somewhat intricate (yet generic and actively generated), is not listed.

#### Object Embedding

In Figure 5 (a subdiagram of Figure 4) you can see the symbiotic relationship between the wrapper instances and the JDO-layer instances. The business model views a Man instance as an entity composed of two wrapper fields, Name and Age, that respectively reference a String object and an Integer object. The JDO model, being less abstract – in order to be efficiently persisted (Goal B) – has no explicit Name or Age objects. Instead, a ManJdo instance shares the same String/Integer instances with its ManImpl wrapper. The following code fragment (line numbers correspond to tag numbers in Figure 6) illustrates this sharing. The code creates a man and assigns it a name. Figure 6 illustrates the procedure, marking each element with the code line that created it. Note that, when updated, the ManImpl wrapper internally synchronizes its ManJdo accordingly:

```

1 public static void main(String[] args) {
2   Man man = new ManImpl();           // implicit creation
  of ManJdo
3   Name name = new NameImpl("John Smith");
4   man.setName(name);                 // implicit update
  of ManJdo
5 }

```

Not surprisingly, we call this feature *embedding*, since the persisted ManJdo class actually embeds fields pertaining to several wrapper objects (e.g., NameImpl, AgeImpl). In fact, we use our metadata repository to define which fields in an entity are embedded and which are simply referenced. Our framework also supports (to some extent) the embedding of "complex" types, themselves containing multiple types, though the concrete implementation details would be out of scope. For instance, an Address composed of street, city, zip code, state, and country can be declared embedded in a Person. And while the business logic would work in terms of person and address interfaces, the only class to be actually persisted would be PersonJdo.

#### Delta Support and Change Tracking

Our project's functional requirements turn delta support (Goal C) and change tracking (Goal D) into important techni-

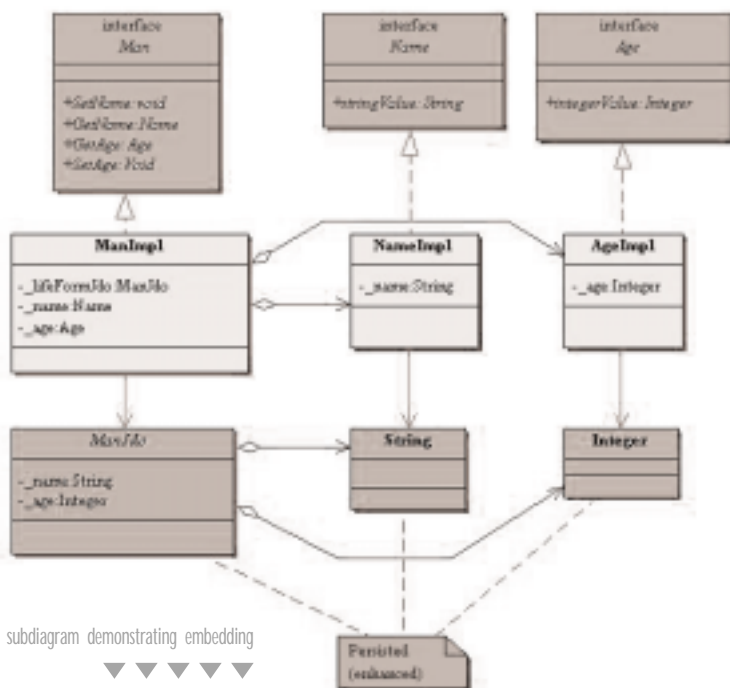


FIGURE 5 Class subdiagram demonstrating embedding

# spiritsoft

[www.spiritsoft.com](http://www.spiritsoft.com)

cal mechanisms. However, given that they're not the principal focal point of this article, we offer the following concise review. (A thorough investigation of these mechanisms would require an article of its own.)

### Delta Support

Supporting remote clients that work on local copies of data entities, we let our client-side code build and transmit an "entity-delta" (the entity subgraph defined only by those fields that have actually been made dirty) to the server. The server-side business logic can now pursue "delta-driven" (i.e., "change-driven") semantics. So, for example, a typical session bean method would fetch the matching persistent entity, apply the delta, or perform some additional logic depending on the delta values, and commit (see Listing 5 for an example of this).

As Listing 5 shows, each entity interface (implementation) has a matching – actively generated – delta interface (implementation) exposing only the getter (accessor) methods. This is both type-safe and logically firm.

Our entities (in fact, the wrapper objects) support the creation and application of deltas by continuously keeping track of the changes made on them during the transaction. The delta-build and delta-apply code is entity-specific – i.e., it's actively generated along with each data entity. We chose this method over a general mechanism that uses reflection due to complexity, maintainability, and speed issues.

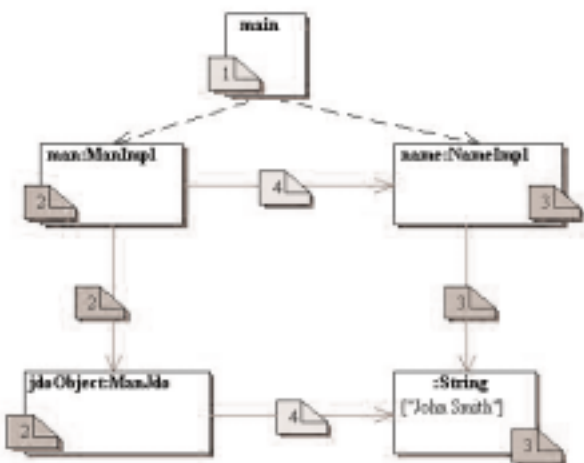


FIGURE 6 Instance diagram demonstrating embedding

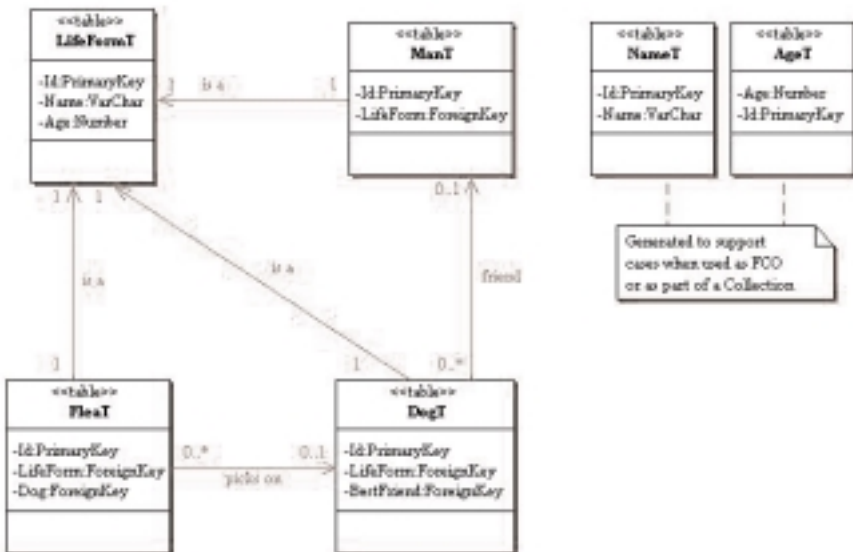


FIGURE 7 Database schema after "smart embeddings"

### Change Tracking

Our server-side framework often needs to be handed the entities changed so far (created, updated, or deleted) during the transaction. It needs these in order to perform certain generic functional mechanisms. The persistence framework helps out by supplying a ChangeTracker component, which cooperates with the wrapper layer in tracking these entities (JDO provides instance callbacks that help maintain this information).

### JDO and Database Layers

The JDO layer is made up of the classes that are actually persisted (enhanced), i.e., the JDO-suffixed classes (see Figure 4). JDO regards dates, strings, and Java primitive wrappers like Integer and Float as fundamental SCOs. Consequently, the typical vendor will choose to embed such fields in their owning entity's table. In our case, since the enhanced LifeFormJDO contains fields referencing a String and an Integer, the correspondent LifeFormT table directly contains the Name and Age columns. We can thus see that, due to the embeddings performed by their wrappers, the JDO layer classes – and so too the conforming database tables – are better suited to efficient O/R mappings. The database schema depicted in Figure 7 conforms to the JDO layer classes of the life form example.

### Performance Analysis (Round 2)

If you compare Figure 7 with Figure 2, you can see that the table structure has improved since Round 1. It has become more compact, resulting in faster, more efficient runtime behavior. Fetching a LifeForm requires only a single fetch as opposed to three fetches plus two joins, as required by the naive implementation. When looking at the toy example, the improvements might seem minor. But the framework's smart-embedding capabilities become much more important when you're confronted with large data sets and a variety of entity and field types, as is the case with e-businesses.

### Conclusion

Summing up, the JDO specification is a promising newcomer in the object persistency arena, even in light of its current shortcomings. Its pure semantics and light model seamlessly integrate within the J2EE environment, making it a viable candidate for small as well as large J2EE projects. However, on projects that involve a rich set of interfaces, classes, and types, in addition to large runtime data sets, developers should take care when building their persistency layer around a JDO product to ensure abstraction and performance.

In this article we've tried to contribute from our experience of doing just that by presenting the JDO-based framework we've built for our own project. We demonstrated how our framework attains the important added-value goals of abstraction and performance, while providing additional capabilities such as code generation, delta-oriented business logics, and more. ●

### Resources

- Java Data Objects Public Access (JDO official site contains JDO specification): <http://access1.sun.com/jdo>
- Rettig, M.J., with Fowler, M. (2001). *Reflection vs Code Generation*: [www.javaworld.com/javaworld/jw-11-2001/jw-1102-codegen\\_p.html](http://www.javaworld.com/javaworld/jw-11-2001/jw-1102-codegen_p.html)
- J2EE specification: <http://java.sun.com/j2ee/>
- The infamous "Java Data Objects vs Entity Beans" duel: [www.theserverside.com/discussion/thread.jsp?thread\\_id=771](http://www.theserverside.com/discussion/thread.jsp?thread_id=771)

# rational

[www.rational.com](http://www.rational.com)

**Listing 1: LifeForm entities – naive implementation**

```
// LifeForm.java
public abstract class LifeForm {
    // private members
    private Age _age;
    private Name _name;

    // methods
    public Name getName() {
        return _name;
    }
    public void setName(Name name) {
        _name = name;
    }
    public Age getAge() {
        return _age;
    }
    public void setAge(Age age) {
        _age = age;
    }
}

// Man.java
public class Man extends LifeForm {
}

// Flea.java
public class Flea extends LifeForm{
}

// Dog.java
public class Dog extends LifeForm {
    // private members
    private Collection _fleas = new Vector();
    // Conveniently enough, we initialize the Fleas collection in
    // the Dog
    // constructor, thus a Dog with no Fleas (assuming one exists?)
    // will hold
    // an empty Collection
    private Man _bestFriend;

    // methods
    public Collection getFleas() {
        return _fleas;
    }
    public Man getBestFriend() {
        return _bestFriend;
    }
    public void setBestFriend(Man bestFriend) {
        _bestFriend = bestFriend;
    }
}

// Name.java
public class Name {
    // private members
    private String _name;

    // constructor
    public Name(String name) {
        if (!validate(name)) {
            throw new IllegalArgumentException(
                "Invalid value: name must contain only letters");
        }
        _name = name;
    }

    // methods
    public String stringValue() {
        return _name;
    }
    private boolean validate(String name) {
        for (int i = 0 ; i < name.length() ; i++) {
            if (!Character.isLetter(name.charAt(i)))
                return false;
        }
        return true;
    }
}

// Age.java
public class Age {
    // private members
    private Integer _age;

    // constructor
    public Age(Integer age) {
        if (age.intValue() < 0) {
            throw new IllegalArgumentException(
                "Invalid value: age must be a positive integer");
        }
        _age = age;
    }

    // methods
    public int intValue() {
        return _age.intValue();
    }
}
```

**Listing 2: XML file used by JDO enhancer**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jdo SYSTEM "jdo.dtd">
<jdo>
```

```
<package name="lifeforms">
  <class name="Name"/>
  <class name="Age"/>
  <class name="LifeForm">
    <field name="_name" embedded="true"/>
    <field name="_age" embedded="true"/>
  </class>
  <class name="Man" persistence-capable-superclass="LifeForm"/>
  <class name="Flea" persistence-capable-superclass="LifeForm"/>
  <class name="Dog" persistence-capable-superclass="LifeForm">
    <field name="_fleas">
      <collection element-type="Flea" embedded-element="true"/>
    </field>
  </class>
</package>
</jdo>
```

**Listing 3: Session Bean working on life forms**

```
public class DogServices implements SessionBean {
    // A CMT stateless session bean

    public void createDog(Age age, Name name,
        Collection fleas, Man bestFriend) {
        PersistenceMnager pm = JDOFactory.getPersistenceManager();
        Dog dog = new Dog();
        dog.setAge(age);
        dog.setName(name);
        dog.getFleas().addAll(fleas);
        dog.setBestFriend(bestFriend);
        pm.makePersistent(dog);
    }

    public void addFlea(Object dogId, Flea flea) {
        PersistenceMnager pm = JDOFactory.getPersistenceManager();
        Dog dog = pm.getObjectById(dogId);
        dog.getFleas().add(flea);
    }

    public void deleteDog(Dog dog) {
        PersistenceMnager pm = JDOFactory.getPersistenceManager();
        pm.deletePersistent(dog);
    }

    // ...
}
```

**Listing 4: LifeForm interfaces**

```
// LifeForm.java
public interface LifeForm {
    public Age getAge();
    public void setAge(Age age);

    public Name getName();
    public void setName(Name name);
}

// Age.java
public interface Age {
    Integer integerValue();
}

// Name.java
public interface Name {
    String stringValue();
}

// Flea.java
public interface Flea extends LifeForm{
}

// Man.java
public interface Man extends LifeForm {
}

// Dog.java
public interface Dog extends LifeForm {
    public Collection getFleas();
    public Man getBestFriend();
    public void setBestFriend(Man man);
}
```

**Listing 5: Sample code, using Delta**

```
// Client side code
public void someFunc(Dog dog) {
    ...
    dog.setAge(new AgeImpl(7));
    dog.getFleas().remove(dog.getFleas().iterator().next());
    DogServices services = ServiceFactory.getDogServices();
    // send dog's delta to server
    services.updateDog(dog.buildDelta());
    ...
}

// Server side code
public void updateDog(DogDelta dd) {
    ...
    PersistenceMnager pm = JDOFactory.getPersistenceManager();
    Object oid = dd.getObjectId();
    Dog dog = pm.getObjectById(oid);
    if (dd.getAge() != null)
        dog.applyDelta(dd);
    else ...
    ...
}
```

# altova

[www.altova.com](http://www.altova.com)



KEITH BROWN J2SE EDITOR

# To Be or Not To Be Certified...

Yesterday I received an A4 piece of cardboard from Sun Microsystems signed by Scott McNealy. I peered at the signature and angled it to the light to see if it was a printed signature or a real one from the pen of Mr. McNealy himself. It was hard to tell.

The piece of cardboard said that I had "Fulfilled all requirements as a Sun Certified Programmer for the Java 2 Platform." Hooray! All those nights trawling through an unbelievably thick book had paid off. Not to mention the time and money invested in a five-day Java course from Sun Educational Services. I stopped short of buying the mock exam from Sun, figuring I had already contributed enough to the coffers of their education division.

I sat for the exam just before Christmas in a small room at the top of an office block in Edinburgh on a dreary Monday morning. Along with five other candidates, who were doing various other exams such as Cisco and Microsoft, I sat in front of a slightly dated PC, clicked Start, and it was straight into question 1 of 59.

After an hour or so, it was complete and I was pleasantly surprised to see 81% pop up on the screen once I had clicked the dreaded Submit button. 81%! Unheard of! I hadn't achieved more than 70% in any of the mock exams I had taken.

After the euphoria had died down (about two days later) I started thinking about the whole validity of certification exams and my own motivation for enduring the process. I use the word *enduring* on purpose because it was not an enjoyable experience. It was like going back to school, memorizing pieces of information that seem totally and completely irrelevant, such as remembering all the adapter

classes in the AWT API. Now, about one month after the event, I've forgotten a lot of these facts because, frankly, they're irrelevant.

What does it really say about a programmer's skills? Does it say "He must be a good programmer because he's Sun certified?" I don't think so. In fact I'm fairly convinced that someone with no Java experience at all could, with the help of one of those big fat "get certified" books, pass the exam. It's similar to any other exam - it's a question of remembering certain facts that the examiner requires you to remember. My programming skills have not been enhanced by this process. I may be able to reel off the main methods of the `java.lang.Math` class without looking at the documentation, but does that make me a better programmer?

Why did I do it? I did it because I'm playing the game. I'm playing the game invented by Sun (and all other companies that offer certification exams) and played by employees and employers. I don't think employers are naive enough to think that "Sun certified" on a résumé means "great programmer." What it does say, however, is that this person has made an effort to become better, to improve his or her knowledge, and is willing to work.

I'm glad I did it. It gives me a sense of confidence and achievement. Also, I got a wee badge that says "Sun Certified Programmer" that I can wear to annoy my colleagues and listen to their inevitable mocking.

To get some real Java programming credentials under your belt, I would suggest that the next stage of the exam process, the Sun Certified Developer for Java 2 Platform, would be a worthwhile

To Be or Not To Be Certified...  
Should you play the game?  
by Keith Brown

32

Use XML Inside Existing Applications...  
...without learning the XML native programming models  
by John Goodson

34

Pattern Foundations:  
The Open-Closed Principle  
Create systems that are easily maintained and flexible  
by Kirk Knoernschild

40

Test First, Code Later  
A guide to integrating JUnit into the development process  
by Thomas Hammell and Robert Nettleton

48

Dynamic Code Generation  
Dynamic behavior without a large performance penalty  
by Norman Richards

54

Under the Hood:  
`java.lang.reflect.Proxy`  
Finding the information you need  
by Paul McLachlan

64

exercise. You actually have to write some code. The prerequisite for this is the programmer's exam, so in a way, it was a necessary evil.

After further examination, it seems Scott McNealy's signature on my certificate is printed and not lovingly and personally applied. Oh well. At least my résumé looks slightly more convincing than a month ago.

## Test Resources

### Books

- Roberts, S., Heller, P., and Ernest, M. (1999). *Complete Java 2 Certification Study Guide*. Sybex.
- Jaworski, J. (1999). *Java 2 Certification Training Guide*. New Riders Publishing.

### Web Sites

- [www.javaranch.com](http://www.javaranch.com)
- <http://joppa.appliedreasoning.com/javaCert/html/JavaCertification.html> (it's harder than the exam) ☺

keith.brown@sys-con.com

### AUTHOR BIO

Keith Brown has been involved with Java for many years. When he's not coding up client solutions for a European Java company, he can be found lurking in the corridors of conferences all around the world.



# ashnasoft

[www.ashnasoft.com](http://www.ashnasoft.com)

# Use XML Inside Existing Applications. . .

. . .without learning the XML native programming models



WRITTEN BY  
JOHN GOODSON

**B**y now, everyone knows that XML is all about data. Unfortunately, that's about all most people know about it. Depending on whom you talk to, XML is projected to be the framework for replacing all software currently in existence or it's seen as an interesting niche technology.

The future of XML is certainly debatable, but it's clear that XML is a "hot" standards-based technology for defining the interchange of data. One of the problems faced by many companies is that the people in the company who deal with data are used to programming to DBCs - JDBC and ODBC. The XML native programming models - XSLT, DOM, SAX, and JAXP - all sound like formidable hurdles to clear in order to take advantage of XML. This article provides you with some clear guidelines on how to quickly extend existing Java applications to leverage XML using the data access standards you're familiar with, without learning any of the native XML programming models.

## Can Our Applications Integrate with XML?

How many times have we all been asked if our existing Java applications can integrate with incoming XML documents over HTTP? We know we can make our applications work with any relational database that has a JDBC driver, but XML documents are something different. Surprisingly, most, if not all, of the work to accomplish this inside our existing applications has already been done by several JDBC driver vendors. Just as there are JDBC drivers for DB2, Oracle, and Microsoft SQL Server, there are also JDBC drivers for XML documents. Listing 1 provides some JDBC source code for reading in a purchase order from an Oracle table.

This particular purchase order model assumes that someone inside the company has entered order information through an application, and all pertinent data has been saved into a table called ORDERS in our Oracle database. Could we easily expand this model to

also allow us to gather customer orders over HTTP directly from customer sites?

## Our Applications Can Integrate with XML

There are several ways in which we can expand our existing purchase order process to include entering orders from the outside, but let's focus on a methodology in which we scan customer sites and enter orders directly into our ORDERS table. Now the ORDERS table will contain information that someone inside our company entered directly, as well as orders that we've obtained by polling customer sites. Clearly, there are several other ways to obtain this information, but all those methods could be accomplished with similar coding requirements (see Listing 2).

With Listing 2 we expand our existing application by integrating it with dynamic XML content, without writing a single line of DOM, SAX, or XSLT code. Our XML JDBC driver makes the HTTP XML data appear as just another relational database. We can issue SQL SELECT queries against the XML document and program using JDBC methods without understanding the difference between a DOM tree and a SAX event.

## Can We Expose Our Data as XML?

We've shown how to integrate XML documents from outside our company into existing applications using JDBC programming. But how do we expose data as XML for others to consume? Currently, JDBC has no methods to expose relational data as XML; however, there are various ways to do this from a JDBC application without leveraging the JDBC specification and without learning an entire XML API.

For example, suppose we keep our

catalog information internally in some DB2 tables and once a month we send out a printed copy of the items we have available, along with their prices. We could decide to publish the catalog electronically using HTML, but then the information would lose its semantics (a price would appear as a string of characters, not as a money field that could be automatically summed, for example). Instead, we could publish a daily update to our catalog in XML format that could be dynamically downloaded over the Web and leveraged by any standards-based applications (XML-, ODBC-, or JDBC-based) that the company wants to use.

Most every database or JDBC driver vendor offers ways to expose relational data as XML, although many of the mechanisms used require that application developers know quite a lot about XML formatting. Using two of the easier approaches, we'll show examples of how to expose a product catalog that's stored in a relational database into XML format through JDBC.

## We Can Distribute Our Data as XML Documents

Suppose the data we want to expose is stored in a Microsoft SQL Server 2000 database. The code in Listing 3 retrieves the data in XML format, which can then be posted to the corporate Web server for download by external customers or partners.

The FOR XML AUTO clause used in the SQL grammar works only for Microsoft SQL Server. To show a difference in approaches, Listing 4 provides another example of similar functionality that's provided by jXTransformer, a product by DataDirect Technologies that's currently in beta. This second approach works against all databases, not just SQL Server.

**bea**

[www.bea.com](http://www.bea.com)

We exposed our internal relational data externally on the Web via XML. Again, we didn't have to learn much about XML to do this, and we didn't have to do any XML API programming. We used JDBC with SQL grammar changes or a JDBC object wrapper API.

### What About Web Services?

We can read XML data and integrate it with our relational databases; we can store relational data as XML; however, are there better ways to use relational assets within the context of the Web while still leveraging standards-based data access protocols like JDBC? Although many might think the answer is "no," the JDBC RowSet interface provides all the mechanisms necessary to use relational data assets within a Web services environment.

A JDBC RowSet object is essentially a cache of data retrieved from a relational data source. The RowSet object extends the JDBC ResultSet interface so programmers can scroll through data, update data, or do anything that a ResultSet object can do with data. The beauty of the RowSet object is that it contains all the logic necessary to interact with the data it holds without being connected to a data source. That is, the RowSet object can be disconnected and function without maintaining an open JDBC connection to its originating data. In addition, the RowSet interface can be serialized or sent over the network to a remote object while still maintaining all its properties.

There are several different implementations of the RowSet interface, but the one that has the most interest for Web applications is the WebRowSet implementation. It's a RowSet implementation that happens to use HTTP for communication and XML for data storage. When a programmer codes to the JDBC RowSet interfaces and deploys a WebRowSet instance, then any of the underlying features of XML and HTTP can be leveraged.

As an example, we might choose to initially populate a WebRowSet with inventory information stored in a centrally located DB2 system. We use JDBC RowSet methods to read the data, but behind the scenes the data is cached in an XML container. We can then make the inventory cache available to customers, partners, or even internal sites over HTTP via standard RowSet methods. Consumers can then read, update, or even send the data to other sites through standard JDBC calls without having a JDBC driver loaded or a live connection to DB2. The data reflects

information that's stored in a DB2 database far from where you're using it. After making changes, we can send the WebRowSet back over HTTP to the originator, where the data can then be resynchronized inside the DB2 database.

### Standards-Based Data Access APIs and XML Are Compatible

Through the use of some simple techniques with existing standards-based APIs, you can leverage the benefits of XML without learning any new XML APIs. Extending existing applications to the Web might not be as difficult as you first thought. You can integrate

your applications with XML by using an XML JDBC driver to make the HTTP XML data appear as just another relational database. You can distribute data stored in a Microsoft SQL Server 2000 database as XML documents by using the FOR XML AUTO clause in your application. You can use relational data assets within a Web services environment by using the JDBC RowSet interface. And, when you use any of these approaches, you don't have to learn the XML native programming models - XSLT, DOM, SAX, and JAXP - to take advantage of XML. ☛

john.goodson@datadirect-technologies.com

#### Listing 1

```
// Get list of orders placed since yesterday
ResultSet PurchaseOrder = stmt.executeQuery (
    "select CUSTOMERID, ORDERID, QUANTITY, ITEM, EXTRADetail
    from ORDERS
    where ORDERDATE > {fn CURDATE()} - 1");

while (PurchaseOrder.next() ) {
    int CustomerId = PurchaseOrder.getInt ("CUSTOMERID");
    int OrderId = PurchaseOrder.getInt ("ORDERID");
    int Quantity = PurchaseOrder.getInt ("QUANTITY");
    String Item = PurchaseOrder.getString ("ITEM");
    String Item = PurchaseOrder.getString ("EXTRADetail");

    // internal business rules dictate how purchase order is processed
    ...
}
}
```

#### Listing 2

```
// Iterate through customer sites to see if there are orders

// we'll use two JDBC connections ... one for our internal Oracle tables and
// one to the
// XML streaming information from the web

// Oracle ...
Connection Oraclecon = DriverManager.getConnection (OracleDriverURL, uid,
    pwd);
Statement stmt = Oraclecon.createStatement();

// XML ...
Connection XMLCon = DriverManager.getConnection (XMLDriverURL, Xuid, Xpwd);
Statement XMLstmt = XMLCon.createStatement();

// Internally, we'll have an Oracle table called CUSTOMERS that tracks
// all our customers
// as well as from which URL we can obtain their order information ...
// provided they can
// post e-orders

// For Customers that can process orders online ... get the URL where we
// collect order info.
// The URL will have a form similar to
// HTTP://www.acme.com/MyCoorders.asp
ResultSet Customer = stmt.executeQuery (
    "select CUSTOMERID, CUSTOMERURL
    from CUSTOMERS
    where ONLINEORDERS = 'T' ");
```

#### AUTHOR BIO

John Goodson is vice president of product operations for DataDirect Technologies, a supplier of components for connecting data and applications across diverse environments. John leads the product strategy and development efforts for the company's connectivity technologies. He has active memberships in the JDBC Specification Expert Group and Java Rowset Expert Group.

# mongoose

[www.portalstudio.com](http://www.portalstudio.com)

```

while (Customer.next() ) {
    int CustomerId = Customer.getInt ("CUSTOMERID");
    String CustomerURL = Customer.getString ("CUSTOMERURL");

    // see if the customer placed any orders by reading a dynamically
    // generated XML document
    // that contains their order information from their website and,
    // if so,
    // insert the order into our ORDERS table
    ResultSet NewOrder = XMLstmt.executeQuery (
        "select QUANTITY, ITEM, DETAIL from " + CustomerURL +
        "where ORDERDATE > {fn CURDATE()} - 1");

    while (NewOrder.next() ) {

        // customer placed order ... gather data and insert the order into
        // our Oracle ORDERS table
        ...

    }

}

```

#### Listing 3

```

// Build a copy of the catalog as an XML Document that can be
// exposed to customers

// Query for the products currently available and include the FOR
// XML AUTO clause
// which will instruct SQL Server to return the result as an XML
// Document
ResultSet OurCatalog = stmt.executeQuery (
    "select PRODUCTID, DESCRIPTION, PRICE from PRODUCTS
    where PRODUCTFORSALE = 1 FOR XML AUTO");

```

```

while (OurCatalog.next() ) {
    // 3 columns selected but only 1 returned ... the XML Document that
    // corresponds
    // to the result set produced from the query
    String XMLCatalog = OurCatalog.getString (1);

    // the XML Document now sits in our XMLCatalog data structure
    // which can
    // be written to a file

    ...
}

```

#### Listing 4

```

// Build a copy of the catalog as an XML Document that can be
// exposed to customers

// Query for the products currently available, which will result
// in an XML-formatted result
StringBuffer OurCatalog = new StringBuffer();
OurCatalog.append ("select xml_element ('product-id',p.PRODUC-
TID),");
OurCatalog.append (" xml_element('product-description',p.DESCRIP-
TION),");
OurCatalog.append (" xml_element('product-price',p.PRICE) ");
OurCatalog.append (" from PRODUCTS p where p.PRODUCTFORSALE = 1");

// Construct object to execute
JXTRQuery ProductCatalog = new JXTRQuery (conn, new String
(OurCatalog));

// Execute, generate XML, and write to file
...
generateImplicitRoot=true;
ProductCatalog.executeWriter (systemOutWriter,
generateImplicitRoot,systemOutWriter.getEncoding,2);

```


[www.javadevelopersjournal.com](http://www.javadevelopersjournal.com)

# int

# www.int.com

# **inseession**

[www.inseession.com](http://www.inseession.com)

written by Kirk Knoernschild

# Pattern Foundations:

## The Open-Closed Principle

Create systems that are easily maintained and flexible

## Design

patterns exploded onto the scene when the seminal work, *Design Patterns: Elements of Reusable Object-Oriented Software*, was published in 1994. Since that time, numerous books on patterns have been written, conferences devoted solely to the patterns movement have emerged, and entire Web sites are dedicated to discussions on patterns. Compound patterns that represent a combination of patterns have even been discovered.

While the benefits we've realized through this movement should not be questioned, the sheer number of patterns has created a situation in which it has become increasingly difficult to identify the most useful patterns for a particular context. In addition, it's common to find that many patterns have strikingly similar characteristics. When designing an application, I've found that simply determining which pattern to use can be as much of a struggle as actually customizing it to my specific context and implementing it.

As a Java developer and architect, I always try to ensure that my designs are resilient, robust, and maintainable. But equally important as designing flexible systems is designing architecturally simple systems. Because all patterns have consequences, it's important to consider if the advantages of using a pattern to solve a design challenge outweigh the dis-

advantages associated with the additional complexity. This decision, combined with the overwhelming number of patterns from which we have to choose, can paralyze a design effort.

In reality, the usage of a pattern is typically not an up-front decision. Instead, the incorporation of a pattern into our system is an evolution of the system's design because of the need for additional flexibility. We may start with a simple, straightforward solution, only to find that as the system evolves, additional flexibility is required. Based on the situation and the flexibility required, some patterns are better candidates than others.

If this is the case, we must be able to bridge the gap between our initial designs and the resulting patterns we incorporate into our designs. Some underlying principles must exist that we can judiciously apply throughout the entire design process, helping to ensure we create the simplest design possible that offers the maximum amount of flexibility. Were we to examine the structure of a set of patterns, we would find some fundamental principles that reside at their core. If we can understand these principles, it's much more likely that we can not only take better advantage of patterns, but also ensure a more solid design when we're not able to find the most appropriate pattern. Even more so, we may discover our own patterns.



**dice.com**  
www.dice.com

## The Challenge of Objects

Object-orientation has been promising reuse for a long time. As we've come to realize, reusable components are not designed and developed early in the project, but typically grow as the project evolves. For our systems to grow, however, their structure must be resilient and flexible enough so they can undergo iterations of change without compromising their structural integrity.

For a system to exhibit the resiliency we require to allow constant evolution, the relationships existing between our system's classes must be flexible. Flexible dependency management, therefore, is key in establishing our software architecture and should be considered each time we create a new relationship between classes. In fact, were we to examine any arbitrary collection of patterns, we would find that many focus on managing the dependency relationships between classes.

Dependency management facilitates reuse in that it allows us to use the existing behaviors exhibited by a class, while configuring the class with the desired new behaviors. In an extreme case, if we could configure all behaviors of a class with the behaviors we desired, the class would be highly reusable, though it would be of little use because of its high-configuration demands. An example of an extremely flexible and configurable design is the Visitor pattern.

## The Open-Closed Principle

The Open-Closed Principle lies at the heart of the object-oriented paradigm. In a sense, we can say that its the goal of our object-oriented designs. Examining a myriad of patterns reveals its application often. It states the following:

*Software entities should be open for extension, but closed for modification.*

At first glance, this statement seems quite contradictory. How can we possibly create a collection of Java classes so that we can extend the system, adding new functionality, without modifying the existing application classes? Consider a financial institution where we have to accommodate different types of accounts in which individuals can make deposits. In addition, we've found that the algorithm used to deposit information to an account varies significantly depending on the type of account. Figure 1 shows a UML class diagram with accompanying code snippets that illustrates how we might structure a portion of our system. Let's discuss the significant aspects of this design.

### Abstract Coupling

Our Account class has a relationship with our AccountType abstract class. In other words, our Account class is coupled at the abstract level to the AccountType inheritance hierarchy. Because our Savings and Checking classes each inherit from the AccountType class, we know that through dynamic binding we can substitute instances of either of these classes wherever the AccountType class is referenced. Subsequently, Savings and Checking can be freely substituted for AccountType within the Account class. This is the intent of an abstract class, and it's what allows us to effectively comply with OCP. Because Account is not directly coupled to either of the concrete Savings or Checking classes, we can extend the AccountType class, creating a new class such as MoneyMarket without having to modify our Account class. We have achieved OCP compliance and can now extend our system without modifying its existing code base.

## DESIGN PATTERNS

Experienced developers typically have an uncanny ability to create flexible, resilient, and robust software designs. Realizing that many new challenges are simply different flavors of previous challenges, these developers have built an arsenal of best design practices, or patterns, that they apply continuously, often in a slightly different form. Sharing these patterns with others allows the entire development community to realize the benefits of using these proven designs to help create more flexible software.

Formally, a design pattern is a documented solution to a recurring design challenge that's customized given the context of the problem. In addition to describing the solution to a common design challenge, a pattern also discusses the various consequences of using that solution, as well as common implementation techniques. Because patterns can be referenced by name, developers who are knowledgeable about patterns find them to be resilient design solutions as well as effective communication devices when discussing design alternatives.

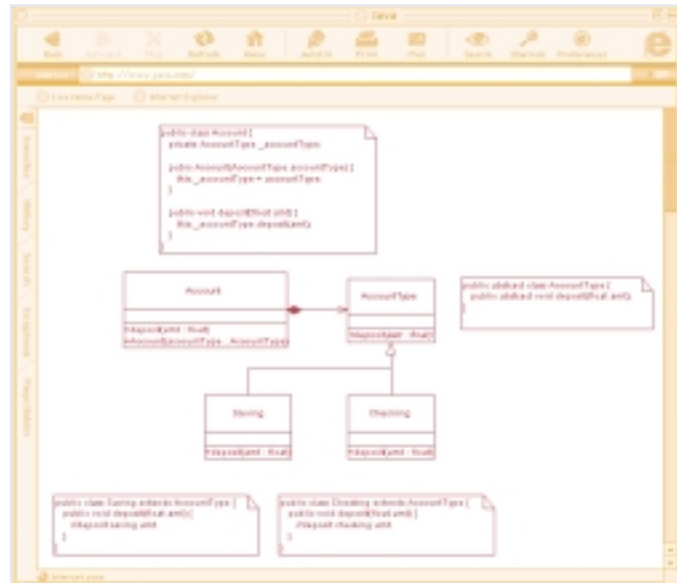


FIGURE 1 Single account class

## Alternative Designs

What is possibly more interesting is how we arrived at the solution in the first place. We know from the requirements that we may need to accommodate new types of accounts in the future. If we didn't know this, our design might be different because we wouldn't need this flexibility and accompanying complexity. Let's first discuss how we arrived at the resulting design seen in Figure 1.

Whenever we consider the use of inheritance, we have a few options, such as:

- **Simple attribute:** This option doesn't utilize inheritance at all. Instead, we forgo the use of inheritance to encapsulate the logic directly in the class. If the value of this attribute, at any point in the life of an object, impacts behavior, we'll find conditional logic present that's based on the value of this attribute. While simple conditional logic may not be cause for concern, as the system grows it's likely that this simple logic becomes more complex and that new behaviors are realized as different attribute values are introduced. This would certainly violate OCP as we would need to modify our existing code base for each new type of account. This is shown in Figure 2, with accompanying Java code snippets.
- **Subclass directly:** In this scenario, we subclass directly based on the variant cases. Subsequently, what was previously a conditional statement when using a simple attri-

**john willy**

[www.willy.com](http://www.willy.com)

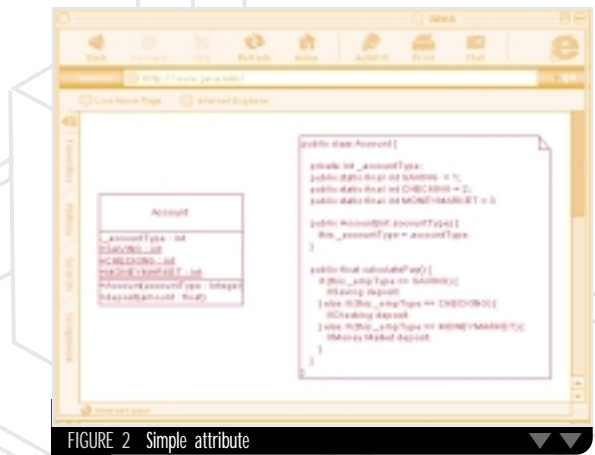


FIGURE 2 Simple attribute



FIGURE 3 Subclass directly

but is now deferred to the individual subclasses. While this approach can help rid the class hierarchy of complex conditional logic, it introduces a different problem. If we find that other attributes impact behavior in a way that's not consistent with our subclasses, we're forced to introduce potentially complex conditional logic based on this attribute, or subclass again. This is shown in Figure 3, with accompanying Java code snippets.

- **Composition:** Using composition, the design challenge is solved using a flexible composite relation, which is the solution we illustrated in Figure 1. While this may be the most complex of the three solutions, it's also likely the most flexible. This scenario also results in the most classes being produced, adding to the complexity.

Let's examine each of the above solutions in the context of the deposit calculation using our Account class. From the requirements we know that deposit is based on the type of account we're dealing with. We'll assume that three different account types – savings, checking, and a newly added money market – each have an algorithmically different calculation for deposit.

**DISCRIMINATOR**

The *discriminator* is a term used to describe the single variant that's the cause for our creation of subclasses. The relationship between any descendant and ancestor should involve only a single discriminator. If we find that any ancestor-descendant relationship has multiple discriminators, we should carefully consider an alternative solution, such as composition. Rarely does subclassing the descendant based on another discriminator result in a flexible enough solution.

We'll begin by examining the solution using a simple attribute. Figure 2 shows an Account class with a single private instance attribute named `_accountType`. The value for this attribute is set within the constructor when the Account instance is created. When the deposit method is invoked, the value of this attribute will be checked and the deposit will be made appropriately, based on the account type.

There are a few disadvantages with this approach. First, new types of accounts or new deposit calculations require a change to our existing Account class. This is an obvious violation of OCP, as the Account class is not closed to modifications. Second, the `_accountType` variable is not type-safe. For instance, if a value of four was passed to the constructor, the Account instance would blindly set the value of `_accountType` equal to the invalid value of four. Of course, we can always put some code in the constructor to ensure that the value passed in is within the appropriate range, but this only adds to the OCP violation.

Now, for new types of accounts or deposit calculations we need to update the deposit method with an additional conditional and make sure we remember to modify the constructor to allow the `_accountType` instance attribute to accommodate this new value within its range of values. As the functionality grows and evolves, the Account class will become unwieldy to work with.

Figure 3 illustrates an equivalent solution using subclasses. Here, we subclass an abstract Account class with three subclasses representing each account type. In this situation we rely on some client to create the appropriate instance of the descendant class directly. This creation might be done in an object factory. In this situation, we no longer have any conditional logic based on the type of account because all this logic is now deferred to the appropriate Account descendant.

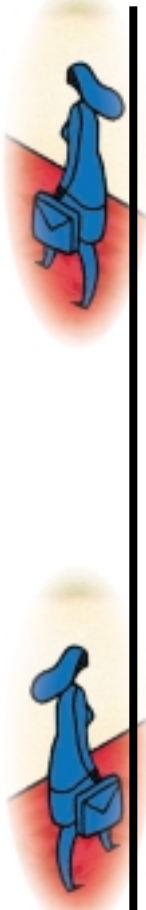
Common attributes and behaviors can easily be implemented on the ancestor Account class. In addition, any class referencing any of the descendants of the Account class in reality holds a reference to the Account class data type directly. Due to the nature of inheritance and dynamic binding, we should be able to substitute descendants anywhere the ancestor is referenced. It appears we have achieved OCP compliance.

In addition, our type safety problem discussed earlier should no longer be cause for concern when subclassing. In this situation, we have only three classes that can be instantiated. We don't have the ability to create an instance of any other type of class that could cause a similar problem.

While a bit more complex, this approach is certainly more flexible than that used in the previous example; however, it has some disadvantages. First, inheritance is a very static solution. If we were to identify additional behavioral variants that didn't fit into the realm of our subclasses, we would be left with a structure that doesn't flex in the direction of our application. For instance, consider a new or changing requirement that impacts the behavior of our Account class based on an account status. Unfortunately, the account status may not fall along the same behavioral lines as the account type. In fact, contriving an example, our account status may fall into one of three categories: active, inactive, or overdrawn. If this new requirement impacts how a deposit is

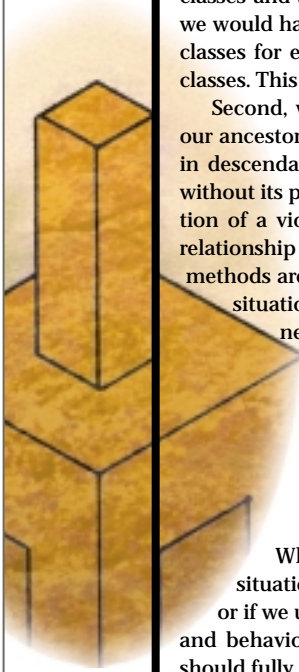
**OBJECT FACTORIES**

Object factories allow us to decouple the instantiation of an object from the object that will actually be using the instance. Using a factory allows us to reference only an ancestor class or interface in our client classes. Subsequently, modifications made to the descendant of an abstract class or interface are encapsulated within the inheritance hierarchy.



# actuate

[www.actuate.com](http://www.actuate.com)



made, or any behavior on the Account class, we might find we're relegated to using a simple attribute with conditionals. Considering this new requirement, let's examine some of our options.

First, we can try subclassing each of our account descendants directly and create separate status classes for each AccountType subclass. This would result in a proliferation of classes and a high probability of duplicate code. In essence, we would have an active, inactive, and overdrawn set of subclasses for each account descendant, resulting in nine total classes. This is extremely inflexible.

Second, we could define some default status behavior in our ancestor Account class that would override this behavior in descendants with variant behavior. This, however, is not without its problems either. Overriding methods is an indication of a violation of the generalization over specialization relationship inheritance is used to represent. While some methods are designed to be overridden, it typically creates a situation where code must be duplicated, especially as new descendants are added.

“ The Open-Closed Principle lies at the heart of the object-oriented paradigm. **In a sense, we can say that it's the goal of our object-oriented designs** ”

While this second approach works well for simple situations, it doesn't scale well as complexity increases, or if we use the Account class to store common attributes and behaviors. Any time we decide to use inheritance, we should fully understand the discriminator that's the cause for the creation of the descendants.

Referring back to our original design in Figure 1, we see the solution resolved using composition. Here, we're back to a single Account class that's composed of an AccountType abstract class. We again see that upon instantiation of the Account class, the AccountType is passed to the constructor. While a bit more complex, this solution is much more flexible. We certainly have the ability to extend AccountType and create new types of accounts without modifying the Account class directly. Our AccountType abstract class serves as a contract between the Account and its descendants.

In addition, if a new requirement is introduced that presents us with the need to incorporate account status into the structure, we can easily create a new AccountStatus abstract class with the appropriate descendants to satisfy status-specific behavior. In other words, our inheritance hierarchies are very small and involve only a single discriminator. This is a much more flexible solution.

To recap, we've discussed three alternatives to deal with the design challenge. Using a simple attribute is the easiest, but it also offers the least flexibility. This approach works well when the attribute does not impact behavior. The remaining two comply with OCP, but each has different advantages. Subclassing works well if we're sure we won't need to introduce a new discriminator. It's been my experience that this approach works well for nondomain-oriented approaches. Typical business domains are usually too dynamic to ever be sure of anything. The composition approach, while the most complex, also offers the greatest flexibility.

**Implementation Challenges**

Should we decide to go with either of the remaining two approaches, there are a few implementation-specific issues

we'll need to address. If the intent of OCP is to ensure we can extend a system without modifying the existing elements, we must make sure the system does not refer to any of our concrete classes directly. This works fine, except for object creation. To create an object, we must refer directly to the class that the object is an instance of. We have two options: to dynamically load the class or use an object factory. While discussion of these topics is beyond the scope of this article, you're encouraged to experiment with each of these approaches.

The fact that we must reference the concrete class explicitly in order to create an instance indicates that an entire system can never be closed against all types of modifications. Therefore, effort should be spent applying OCP in a tactical manner, focusing on the areas of the system that are most dynamic and demand the flexibility of OCP.

**Conclusion**

While patterns contribute positively to the resiliency of many object-oriented designs, the proliferation of patterns often makes it an overwhelming challenge to find the one

most suitable given the current context. However, underlying many patterns is a set of fundamental principles that are pervasive in object-oriented systems. The first of these, the Open-Closed Principle, enables us to create systems that are flexible and more easily maintained, allowing us to evolve these systems gracefully.

In reality, achieving OCP-compliance system-wide is not possible. There will always exist some change that the system is not closed against. Therefore, closure must be applied judiciously to the areas of the system that are most complex and dynamic. Even partial OCP compliance results in more resilient systems. Isolating violations of OCP to specific classes, such as object factories, certainly serves to reduce the overall maintenance efforts.

Quite frequently, striving to apply these principles throughout our systems creates a more natural way to discover the solutions to challenging design decisions that are appropriate candidates for flexible patterns. In this article we've discussed just one of the principles underlying many of the patterns we should be using in everyday development. ●

**References**

- Meyer, B. (1988). *Object-Oriented Software Construction*. Prentice-Hall.
- Martin, R.C. (2000). "Design Principles and Design Patterns." [www.objectmentor.com](http://www.objectmentor.com)
- Gamma, E., et al. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.
- Knoernschild, K. (2002). *Java Design: Objects, UML, and Process*. Addison-Wesley.

**AUTHOR BIO**

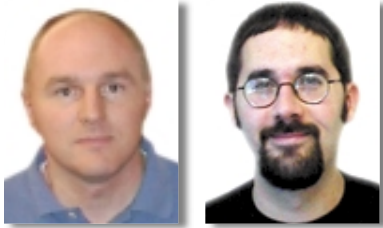
Kirk Knoernschild is a senior consultant and instructor with TeamSoft, Inc. ([www.teamsoftinc.com](http://www.teamsoftinc.com)), a firm that offers development, training, and mentoring services to major corporations. He shares his experiences through courseware development and teaching, writing, and speaking at seminars and conferences. Kirk is the author of *Java Design: Objects, UML, and Process* (Addison-Wesley).

# loox

[www.loox.com](http://www.loox.com)

# Test First, Code Later

A guide to integrating JUnit into the development process



WRITTEN BY  
THOMAS  
HAMMELL &  
ROBERT NETTLETON

**T**esting is usually an afterthought in the development process. The developer's main focus is to design and write code.

Of course, the developer runs the program many times during development to make sure the code runs and produces the expected results; however, this testing has no real structure and the main goal is to ensure the program runs at that moment. Most developers rely too much on QA or the end user to make sure the program works properly and meets requirements.

Extreme Programming has taken the "build a little, test a little" philosophy to a new level by requiring that all classes have unit tests that are run on a regular basis. A unit test is a structured set of tests that exercises the methods of a class. Unit testing is a good idea for the following reasons:

- Gives the developer confidence that the class is working properly
- Quickly finds the source of bugs
- Focuses the developer on the design of a class
- Helps refine the requirements of the individual classes

As unit tests are built for each class, they can be collected and run as a group on a regular basis to track the progress of the project and find bugs as they occur.

Although there are many good reasons to create unit tests, most developers don't believe it's worth the time or don't want to take the time to develop a unit-testing framework. Luckily Erich Gamma and Kent Beck have developed JUnit, a simple unit testing framework that makes developing unit tests almost painless. You can download a copy of JUnit at [www.junit.org](http://www.junit.org).

## The JUnit Framework

JUnit is a set of classes that allows

you to easily create and run a set of unit tests. You use JUnit by extending its classes. Figure 1 shows JUnit's main classes and their relationships.

The central class of JUnit is `TestCase`. `TestCase` represents a fixture that provides the framework to run unit tests and store the results. It implements the `Test` interface and provides the methods to set up the test condition, run tests, collect the results, and tear down the tests once they're complete. The `TestResult` class collects the results of a `TestCase`. The `TestSuite` class collects a set of tests to run. JUnit also provides a command line class (`junit.textui.TestRunner`) and a GUI class (`junit.ui.TestRunner`) that can be used to run the unit test and display the results.

To illustrate how to use JUnit, let's write a simple unit test to test some methods of the Java `String` class. The code, along with a batch file needed to run the unit test, can be found on the **JDJ** Web site, [www.sys-con.com/java/sourcec.cfm](http://www.sys-con.com/java/sourcec.cfm). Listing 1 shows the code for the unit test. The class is called `StringTester` and is a subclass of `TestCase`. The constructor has one argument, name (the name of the unit test being run), that will be used in any messages displayed by JUnit. The next method in the listing is `setUp()`, which is used to initialize a couple of strings.

The next three methods (`testLength()`, `testSubString()`, and `testConcat()`) are the actual tests. The test methods must be no argument methods that test a specific condition using one of JUnit's assert methods. If the conditions of the assert are true, JUnit marks the test as passed, otherwise it's marked as a failed.

The `suite()` method is used by JUnit to create a collection of tests. There are two ways to create a suite:

1. Create an instance of the `TestSuite` and then add the individual tests to the suite.
2. Create an instance of `TestSuite` by passing the test class in the constructor; this creates a suite that contains all the methods in the test class that start with the word `test`. This is the preferred way to create a suite, since it's simpler and helps prevent the accidental omission of tests.

Adding tests individually to a suite is a good way to create a subset of tests to try to isolate a bug. Both JUnit classes used to run the unit tests require the `suite` method to be present.

The class has a main method that can be used to run the unit tests. The main method calls the `junit.textui.TestRunner.run` method to run all the tests of the suite. The main method is useful if the unit tests are run by a batch file. The tests in this class can also be executed using one of the following commands in a console window:

- `java junit.textui.TestRunner StringTester`
- `java junit.ui.TestRunner StringTester`

These assume that the `junit.jar` file is in the classpath. The first command runs the unit tests in command line mode and prints the results to the screen. The second command brings up the JUnit GUI, which shows the results graphically (see Figure 2).

The `StringTester` class has an intentional bug in it to show what happens when a test fails. In the `testLength` class



# **installsheid**

[www.installsheid.com](http://www.installsheid.com)

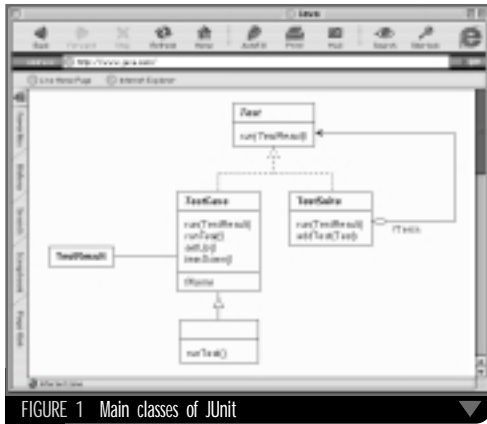


FIGURE 1 Main classes of JUnit

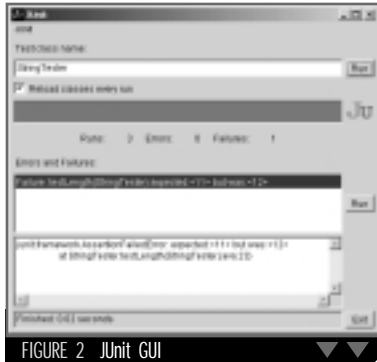


FIGURE 2 JUnit GUI

**AUTHOR BIOS**

Thomas Hammell is a senior developer at Hewlett-Packard/Bluestone Software and is part of the tools group that develops various tools for HP middle-ware products. He has over 15 years of experience developing software. Tom holds a BS in electrical engineering and a MS in computer science from Stevens Institute of Technology.

Robert Nettleton is a software engineer at Hewlett-Packard/Bluestone Software. He currently works as the test lead for HP/Bluestone's Enterprise JavaBean container. He has 5 years of experience developing and testing object-oriented software in C++ and Java. Bob holds a BA in computer science from Rutgers University.

the correct length of the string is really 12, not 11. When a test fails, JUnit reports which test failed and prints out the expected and actual values that failed.

Having an error in the test code brings up a good point. The unit tests should be tested to make sure there are no errors in the test code. The best way to do this is to write the tests first and then stub out the code to be tested. When the unit test is first run, all the tests should fail. Then as the code is filled in, executing the individual unit test helps ensure not only that it passes but also that the test is correct.

That's all there is to writing a unit test with JUnit. The framework is very easy to use. The hardest part is getting used to writing the tests as you write the actual classes. The process of writing unit tests will initially take more time during development, but it will pay great dividends when regression testing needs to be done or a high-risk change needs to be made to the code.

**The Value of Unit Testing in the Real World**

To show the value of JUnit in a real development environment let's look at what a set of completed unit tests for a moderately complex piece of software looks like and how JUnit improves software development.

This software calculates the position of a planet for a given date. There are

four main classes that perform the following functions:

1. **CalcPlanetPosition:** This is the main class used to calculate the position of a planet given a planet number and a date, and it involves three major steps:
  - Calculate the of the number of days between 1/1/1980 and the given date.
  - Calculate the heliocentric coordinates of the planet.
  - Convert the heliocentric coordinates to right ascension and declination.
2. **DateUtils:** Calculates the number of days between 1/0/1900 and the given date
3. **HelioPos:** Calculates the heliocentric coordinates of the planet
4. **ConvertUtils:** Converts the heliocentric coordinates to right ascension and declination

Each class has a corresponding unit test class. The name of the unit test class is the name of the class with the word "Tester" added. The DateUtils class has three main functions. It determines whether the user entered a valid date (isDateValid), calculates the number of days between two dates (calcDateDiff), and calculates the difference between 1/0/1990 and a given date (CalcNumOfDays).

The unit test for the isDateValid method tests a valid date of 8/1/1989 and an obviously invalid date of 13/32/0000 to test basic functionality. It also tests some not so obvious invalid dates like 9/31/1992 and 2/29/2001. For the calcDateDiff method, tests have been written to ensure that the number of days between two dates work when the dates are separated by a day, week, month, and year. There's also a test to ensure that the calculation works in a leap year.

Once the framework is set up it's easy to add new test cases as bugs are found. The complete set of source code, along with the unit tests, is contained in the PlanetPos.jar file and can be downloaded from the *JDJ* Web site

The unit tests for the other classes follow in a similar manner. A special class called SystemTester runs the entire set of tests for the project. The class is a subclass of TestSuite and has a main() and a suite() method. The class creates a test suite that contains an entry for each of the unit test classes.

Once completed, the tests can be run on a regular basis to check for bugs as the development process continues. Once the program is fully working and tested, it's sent to QA where more rigor-

ous testing occurs.

In this example, even though the code was written according to the requirements and passed all the unit tests, there are still two significant bugs that have been found by QA. The first problem is that the calculation of the planet position is not accurate enough for certain planets for certain dates. The second problem is that there's no way to enter dates before 1 AD.

To find the source of the bugs the first step is to write a set of unit tests that reproduces these bugs. First, new test cases are added to test the main class (CalcPlanetPosition) to confirm the results that QA has found. These new test cases will fail, but they accurately communicate the conditions of the bug. The next step is to write new test cases that test the calculations made for the other three classes (DateUtils, HelioPos, and ConvertUtils) for the failure conditions.

At this point the developer may need to consult with the user who wrote the original requirements to determine the pass/fail criteria for the new test cases. Again, the test case serves to communicate the information needed from the user in order to refine the original requirements. Once the new pass/fail criteria are determined, they become new requirements on the system. In this fashion, as bugs are found and features added, the test cases become a set of requirements of higher and higher fidelity that gets tested each time the unit tests are run.

In this particular case it turns out the date format used throughout the project assumes that the dates entered will be after 1 AD. Also, the formula used by calcHelioPosition in the HelioPos class is not good enough to produce accurate results for all cases. A number of methods will have to be rewritten. The rewritten code and unit tests are in the file PlanetPos\_Reworked.jar available on the *JDJ* Web site.

At this point it's late in the development cycle and rewriting these core methods is high risk. But the unit tests that are now written make the risk much lower because once the changes are made, the full set of unit tests can be run and the developer can be confident that he or she has "done no harm."

**Best Practices**

It's important to be methodical and consistent when developing unit tests for a piece of software. The development of a unit test should be treated with the same care that's taken when developing other code. A number of best practices

**ilog**  
[www.ilog.com](http://www.ilog.com)

# “JUnit tests don't take the place of functional testing”

should be employed to keep the unit tests manageable and useful.

## 1. All unit test classes should have a `main()`.

Although the unit tests will normally be run as a group, there are times when it's necessary to run just one of the unit test classes to isolate a bug. Having a main method that just calls the `junit.textui.TestRunner.run` method makes that easy.

## 2. Make sure your test cases are independent.

As the test cases in a unit test are developed, it's important to make sure that each test case can be run independently. This is especially true when testing database applications where the state of the database at the start of a test case must be consistent in order for the test to be effective. It's possible that a test

case will incorrectly handle an exception that occurs during a test run and corrupt the database. This can cause all subsequent test cases to fail. Nothing is more annoying than spending hours debugging an application that appeared to be failing because the test cases were not set up properly.

Don't count on the test cases being run in a certain sequence because the JUnit framework doesn't guarantee the order of test method invocations. The setup and teardown methods are called before and after each test case is run. They can be used to set up a database to a known state before a test is run and restore it to a known state after the test is complete. It's also very important to make sure that test methods make every effort to clean up after themselves in every conceivable exit condition.

## 3. Minimize the amount of code in setup/teardown methods.

Since the setup and teardown methods are called before and after each test case, it's important to make sure they're efficient. Some test suites can contain 50, maybe even 100, test cases. This includes making sure that these methods don't output unnecessary log messages and make it hard to follow the execution path. Making these methods efficient will also speed up the time it takes to run the test, which allows the tests to be run more often.

## 4. Use `fail()` instead of `assert()` to catch test case error.

If the test case fails because of an error or exception as opposed to an `assert` that fails, it's better to use the `Assert.fail(String msg)` than `assert(true,false)`. This will make it easier to wade through the debug statements and can decrease the amount of time required to understand and fix a problem. Using

`Assert.fail(String msg)` with a description of the failure instead of calling `assert(boolean boolValue)` can more directly describe the type of failure occurring. The message passed to `fail(String msg)` will be printed out by the `TestRunner` in the stack track of failures. This allows the developer to quickly see what's really failing in a test case.

## 5. Properly document test code.

As with any code being developed, it's important that the test code be properly documented with Javadocs. Just as it's easier to maintain well-documented code, well-documented test cases are easier to update.

## Things That JUnit Can't Do

JUnit does not separate test data from test code since the data is hard-coded in JUnit. This could be a problem when testing database-driven applications. The JUnit ++ extension solves this problem by creating a test data repository.

JUnit can't unit test Swing GUIs, JSPs, or HTML. But there are JUnit extensions, JFC Unit and HTTP Unit, that can do these things.

JUnit tests don't take the place of functional testing. If all your unit tests pass, it doesn't mean the software is ready to ship. Functional testing still needs to be done to ensure that the software meets the full set of requirements including performance criteria, hardware requirements, and usability.

## Conclusion

Testing should not be an afterthought in the development process. It's a well-known fact that the cost of fixing a bug goes up drastically the later it is found. Unit testing provides a way to find bugs almost as they occur. It also improves the design process and provides an effective way to communicate bugs and requirements between members of the development team. JUnit is an easy-to-use framework that can be used to test any Java class.

Although you may not favor all aspects of Extreme Programming, unit testing will improve any development process and the quality of software produced. ☘

## Reference

- Gamma, E., and Beck, K. "JUnit: A Cook's Tour." <http://junit.sourceforge.net/doc/cookstour/cookstour.htm>

thomas\_hammell@hp.com

robert\_nettleton@hp.com

# solution site

[www.solutionsite.com](http://www.solutionsite.com)

# **intraware**

[www.intraware.com](http://www.intraware.com)

# Dynamic Code Generation



WRITTEN BY  
NORMAN  
RICHARDS

**J**ava is often criticized for its performance, particularly in comparison to equivalent code written in languages such as C and C++. Advances in runtime performance have silenced many critics, but still there are times when no matter how fast Java is, it's just not fast enough.

Fortunately, Java does have some unique properties that make a class of optimizations accessible that are not easily applied to other platforms. One technique that can be used to great effect on certain types of tasks is dynamic code generation.

Dynamic code generation is the process of generating Java classes at runtime that can then be loaded into that same Java Virtual Machine (JVM) and accessed the same way as ordinary compiled Java classes. The usefulness of the technique might not be immediately apparent, but it's actually used to a great degree of success in several critical, high-profile Java technologies.

## JavaServer Pages

Perhaps the best-known example of dynamic code generation in Java is JavaServer Pages (JSP). A servlet engine is capable of dispatching requests to a servlet for processing. Unfortunately, servlets are static in nature. They typically need to be precompiled and pre-configured in the servlet engine before server startup. Although they're quite useful, Web development often requires a more flexible solution.

JSP breaks these restrictions and allows servlets to be dynamically created at runtime based on the contents of a JSP file. When a request is made for a JSP file, the servlet engine makes a request to the JSP engine, which in turn will process the JSP file and send a result. A JSP is a textual representation of a set of actions to be performed to present a page to the user, and interpreting it anew for each request would be quite costly. Instead, the JSP engine will compile the JSP and dynamically create a servlet object. If the specification changes, a new servlet will be dynamically generated.

Dynamic code generation is a big win here because it brings dynamic

behavior without a large performance penalty. The behavior of the runtime system doesn't need to be specified completely at compile time or even at the time the system starts. Yet, at the same time, we avoid the slothful response times that might be associated with interpreting the JSP file for each request.

## XSLTC

A lesser known but more recent example of dynamic code generation is XSLTC. A subproject of the Apache XALAN project, XSLTC is a compiler for XSL stylesheets that can compile a stylesheet into a Java class to perform that specified transformation. XSLTC can function as a traditional compiler or can compile a stylesheet in memory to immediately process an XML input file with the full performance benefit of compiled code.

The power of this technique should not be overlooked. The flexibility of an XML-based specification (which is not necessarily available until runtime) is combined with the performance of executing Java code directly.

## Methods for Code Generation

Now that we've seen two systems that make use of dynamic code generation, let's explore the code generation techniques of both systems by applying them to a simple problem domain. We'll develop a simple, four-function, stack-based integer expression evaluator with variable substitution.

Our expression evaluator will evaluate mathematical expressions such as "4 \$0 + \$1 \*". Stack-based processing of this expression is trivial. We push 4 and variable 0 on the stack and then add the top two values on the stack, replacing both of the operands with the result of the addition. Next we push variable 1 on the stack and perform a multiple operation.

## Dynamic behavior without a large performance penalty

In more familiar algebraic notation, this would be represented as "(4 + \$0) \* \$1". If we were to evaluate this expression with the variable list of "[3, 6]", the result would be (4 + 3) \* 6 = 42.

All of our expression evaluators will conform to the calculator interface. The calculator interface contains a single method, evaluate, that takes an array of integer arguments as input and returns the integer result of performing whatever calculation is required.

```
// Calculator.java
public interface Calculator
{
    int evaluate(int[] arguments);
}
```

## A Simple Expression Evaluator

For demonstration purposes, we'll consider a simple but inefficient expression interpreter that uses a Stack object to evaluate the expression (see Listing 1). The expression is parsed once for each evaluation. However, we will tokenize the expression only once, at object creation time, to avoid the overhead of the StringTokenizer class.

As you can see from the performance numbers, this method is quite slow. It may be adequate for evaluating an occasional expression, but we can do much better.

## Parsing the Expression

If the expression is likely to be reused, a better technique would be to parse the expression and apply the Composite design pattern to create an expression tree (see Listing 2). The internal structure of the tree represents the logic of the expression to be evaluated.

The performance of the parsed calculator is an order of magnitude better than the simple interpreted version above. While there's still plenty of room for optimization, we're not far from the

# parasoft

[www.parasoft.com](http://www.parasoft.com)

# “The ultimate solution would retain the runtime performance of compilation without incurring the cost of invoking an external compiler”

limits of what can be achieved without getting closer to the JVM.

## Compiling the Expression

To make that step closer to the JVM, we'll attempt direct compilation of our expression. We'll generate a Java source file from our expression (in the same way a JSP engine would generate a Java source file) and then compile and load it.

Translating our mathematical expression into a Java expression isn't difficult. “\$0 \$1 \$2 \* +” can be represented by the Java expression “args[0] + (args[1] \* args[2])”. We need to choose a unique name for our class and write the generated code in a temporary file. The generated code for this expression would resemble the following:

```
public class [CLASSNAME]
    implements Calculator
{
    public int evaluate(int[] args) {
        return args[0] + (args[1] *
            args[2]);
    }
}
```

Next, we need to compile the Java code. Unfortunately, we can't count on any particular compiler to be installed on our machine. For this example we'll assume the javac compiler is available and is in our system PATH. If javac is not in our PATH or another compiler is desired, that can be specified with the “calc.compiler” property. For example, “-Dcalc.compiler=jikes”.

If a compiler other than javac is used, we will almost always need to place the Java runtime JAR (rt.jar in the jre/lib directory) in the classpath passed to the compiler. We'll use the “calc.classpath” property to indicate additional classpath elements to be passed to the compiler. For example, “-Dcalc.classpath=/usr/local/java/jre/lib/rt.jar”.

The compiler can be invoked as an external process using Runtime.exec(String[] cmd), which produces a Process object. The elements of the cmd

array contain the system command to be executed. The first element should be the name of the program to execute. The remaining elements are the individual arguments to be passed to the compiler.

```
String[] cmd = {
    _compiler,
    "-classpath",
    _classpath,
    javafilename.getName()
};
```

```
Process process = Runtime.get
    Runtime().exec(cmd);
```

We need to wait for the compile process to end and then grab the return code from the compiler. A process return code of zero indicates a successful compile.

```
process.waitFor();
int val = process.exitValue();
if (val != 0) {
    throw new RuntimeException(
        "compile error. " +
        "compiler return code is " +
        val);
}
```

One final issue with compilation is that it's important to read the output and error streams of the external process. If the compiler generates enough output, the compile process could potentially block waiting for the output to be read. We'll ignore the issue here, but this should definitely be addressed in a production system.

If the compile was successful, we now have a Java class file in the current directory that can be read in. Our ClassLoader needs a byte array to load, so we read the contents into the byte array and ask the system to create a class for us.

```
byte[] buf = readBytes(classname +
    ".class");
Class c = defineClass(buf, 0,
    buf.length);
```

Our ClassLoader is the minimal ClassLoader necessary to accomplish our task. It doesn't perform all the tasks that a production-level ClassLoader would. However, it serves for demonstration purposes.

Once the class is successfully loaded, we can create an instance of the class and return it for use.

```
return (Calculator)
    c.newInstance();
```

The complete code is shown in Listing 3. The runtime performance of the compiled calculator represents another order of magnitude speed improvement over our first attempt. Our same 1,000,000 evaluations run in 100–200ms instead of 1–2 seconds as in the prior example. But there's still a big performance hit in compiling the code. Invoking javac to compile the code can take 1–2 seconds, negating our performance gain. However, javac is not a very high-performance compiler. When we switch to a fast compiler such as jikes, compile time drops significantly, to 100–200ms.

## Generating Bytecode Directly

The ultimate solution would retain the runtime performance of compilation without incurring the cost of invoking an external compiler. An in-memory compiler would be one option, but we can also eliminate the overhead and messiness of compilation by generating the Java bytecode in memory.

While writing Java to a file is relatively simple, the Java class file format is tricky enough that we'll want to use an external bytecode library to generate the file. For this article we use BCEL, the Bytecode Engineering Library.

Using BCEL, we first need to create a ClassGen object to represent our class. As with the compiled example, we have to define a unique classname. Unlike Java code, we need to directly declare our super class of Java.lang.Object. ACC\_PUBLIC specifies that our class is public. We also need to set the ACC\_SUPER access flag that all Java 1.0.2 or higher Java classes should set. Finally, we specify that our class is to implement the Calculator interface.

```
ClassGen classgen =
    new ClassGen(classname,
        "java.lang.Object",
        "",
        Constants.ACC_PUBLIC |
            Constants.ACC_SUPER,
        new String[]
            {"Calculator"});
```



# quintessence

[www.inj2.com](http://www.inj2.com)



# pointbase

[www.pointbase.com](http://www.pointbase.com)

### Listing 1: SimpleCalculator.java

```
import java.util.ArrayList;
import java.util.Stack;
import java.util.StringTokenizer;

public class SimpleCalculator
    implements Calculator
{
    String[] _toks;

    public SimpleCalculator(String expression) {
        ArrayList list = new ArrayList();
        StringTokenizer tokenizer =
            new StringTokenizer(expression);

        while (tokenizer.hasMoreTokens()) {
            list.add(tokenizer.nextToken());
        }

        _toks = (String[]) list.toArray(
            new String[list.size()]);
    }

    public int evaluate(int[] args)
    {
        Stack stack = new Stack();

        for (int i=0; i < _toks.length; i++) {
            String tok = _toks[i];

            if (tok.startsWith("$")) {
                int varnum = Integer.parseInt(tok.substring(1));
                stack.push(new Integer(args[varnum]));
            } else {
                char opchar = tok.charAt(0);
                int op = "+-*/".indexOf(opchar);

                if (op == -1) {
                    stack.push(Integer.valueOf(tok));
                } else {
                    int arg2 = ((Integer) stack.pop()).intValue();
                    int arg1 = ((Integer) stack.pop()).intValue();

                    switch(op) {
                        case 0:
                            stack.push(new Integer(arg1 + arg2));
                            break;

                        case 1:
                            stack.push(new Integer(arg1 - arg2));
                            break;

                        case 2:
                            stack.push(new Integer(arg1 * arg2));
                            break;

                        case 3:
                            stack.push(new Integer(arg1 / arg2));
                            break;

                        default:
                            throw new RuntimeException(
                                "invalid operator: " + tok);
                    }
                }
            }
        }

        return ((Integer) stack.pop()).intValue();
    }
}
```

### Listing 2: CalculatorParser.java

```
import java.util.Stack;
import java.util.StringTokenizer;

public class CalculatorParser
{
    public Calculator parse(String expression)
    {
        Stack stack = new Stack();
        StringTokenizer toks =
            new StringTokenizer(expression);

        while (toks.hasMoreTokens()) {
            String tok = toks.nextToken();

            if (tok.startsWith("$")) {
                int varnum = Integer.parseInt(tok.substring(1));
                stack.push(new VariableValue(varnum));
            } else {
                int op = "+-*/".indexOf(tok.charAt(0));

                if (op == -1) {
                    int val = Integer.parseInt(tok);
                    stack.push(new ConstantValue(val));
                } else {
                    Calculator node2 =
                        (Calculator) stack.pop();
                    Calculator node1 =
                        (Calculator) stack.pop();

                    stack.push(new Operation(tok.charAt(0),
                        node1, node2));
                }
            }
        }

        return (Calculator) stack.pop();
    }

    static class ConstantValue
        implements Calculator
    {
        private int _value;

        ConstantValue(int value) {
            _value = value;
        }
    }
}
```

```
public int evaluate(int[] args) {
    return _value;
}

static class VariableValue
    implements Calculator
{
    private int _varnum;

    VariableValue(int varnum) {
        _varnum = varnum;
    }

    public int evaluate(int[] args) {
        return args[_varnum];
    }
}

static class Operation
    implements Calculator
{
    char _op;
    Calculator _arg1;
    Calculator _arg2;

    Operation(char op, Calculator arg1,
        Calculator arg2)
    {
        _op = op;
        _arg1 = arg1;
        _arg2 = arg2;
    }

    public int evaluate(int args[]) {
        int val1 = _arg1.evaluate(args);
        int val2 = _arg2.evaluate(args);

        if (_op == '+') {
            return val1 + val2;
        } else if (_op == '-') {
            return val1 - val2;
        } else if (_op == '*') {
            return val1 * val2;
        } else if (_op == '/') {
            return val1 / val2;
        } else {
            throw new RuntimeException(
                "invalid operator: " + _op);
        }
    }
}
```

### Listing 3: CalculatorCompiler.java

```
import java.util.Stack;
import java.util.StringTokenizer;

import java.io.*;

public class CalculatorCompiler
    extends ClassLoader
{
    String _compiler;
    String _classpath;

    public CalculatorCompiler() {
        super(ClassLoader.getSystemClassLoader());

        _compiler = System.getProperty(
            "calc.compiler");
        if (_compiler == null) {
            _compiler = "javac";
        }

        _classpath = ".";
        String extraclasspath =
            System.getProperty("calc.classpath");

        if (extraclasspath != null) {
            _classpath = _classpath +
                System.getProperty("path.separator") +
                extraclasspath;
        }
    }

    public Calculator compile(String expression)
    {
        String jtext = javaExpression(expression);

        String filename = "";
        String classname = "";

        try {
            File javafile = File.createTempFile(
                "compiled_", ".java", new File("."));

            filename = javafile.getName();
            classname = filename.substring(0,
                filename.lastIndexOf("."));

            generateJavaFile(javafile, classname,
                expression);
            invokeCompiler(javafile);

            byte[] buf = readBytes(classname + ".class");
            Class c = defineClass(buf, 0, buf.length);
            try {
                return (Calculator) c.newInstance();
            } catch (IllegalAccessException e) {
                throw new RuntimeException(e.getMessage());
            } catch (InstantiationException e) {
                throw new RuntimeException(e.getMessage());
            }
        } catch (IOException e) {
            throw new RuntimeException(e.getMessage());
        }
    }

    void generateJavaFile(File javafile,
        String classname, String expression)
        throws IOException
    {
        FileOutputStream out =

```

# **compoze**

[www.compoze.com](http://www.compoze.com)

```
new FileOutputStream(javafile);

String text = "public class " + classname +
" implements Calculator {" +
" public int evaluate(int[] args) {" +
" " + javaExpression(expression) +
"};" +
"}";

out.write(text.getBytes());
out.close();

void invokeCompiler(File javafile)
throws IOException
{
String[] cmd = {_compiler, "-classpath",
_classpath, javafile.getName()};
Process process = Runtime.getRuntime()
.exec(cmd);

try {
process.waitFor();
} catch (InterruptedException e) {

}

int val = process.exitValue();
if (val != 0) {
throw new RuntimeException(
"compile error: " +
"compiler return code is " + val);
}

byte[] readBytes(String filename)
throws IOException
{
File classfile = new File(filename);
byte[] buf =
new byte[(int) classfile.length()];

FileInputStream in =
new FileInputStream(classfile);
in.read(buf);
in.close();

return buf;
}

String javaExpression(String expression)
{
Stack stack = new Stack();
StringTokenizer toks =
new StringTokenizer(expression);

while (toks.hasMoreTokens()) {
String tok = toks.nextToken();

if (tok.startsWith("$") {
stack.push("args[" +
Integer.parseInt(tok.substring(1)) +

```

```

]");
} else {
int op = "+-*/".indexOf(tok.charAt(0));
if (op == -1) {
stack.push(tok);
} else {
String arg2 = (String) stack.pop();
String arg1 = (String) stack.pop();

stack.push("(" + arg1 + " " +
tok.charAt(0) + " " + arg2 + ")");
}
}
}
return "return " + (String) stack.pop() + ";"
}
}


```

**Listing 4: CalculatorGenerator.java**

```
import java.io.*;

import java.util.Stack;
import java.util.StringTokenizer;

import de.fub.bytecode.classfile.*;
import de.fub.bytecode.generic.*;
import de.fub.bytecode.Constants;

public class CalculatorGenerator
extends ClassLoader
{
public Calculator generate(String expression)
{
String classname = "Calc." +
System.currentTimeMillis();
ClassGen classgen = new ClassGen(classname,
"java.lang.Object", "",
Constants.ACC_PUBLIC | Constants.ACC_SUPER,
new String[] {"Calculator"});

classgen.addEmptyConstructor(
Constants.ACC_PUBLIC);
addEvalMethod(classgen, expression);

byte[] data = classgen.getJavaClass()
.getBytes();
Class c = defineClass(data, 0, data.length);

try {
return (Calculator) c.newInstance();
} catch (IllegalAccessException e) {
throw new RuntimeException(e.getMessage());
} catch (InstantiationException e) {
throw new RuntimeException(e.getMessage());
}
}

private void addEvalMethod(ClassGen classgen,
String expression)
{
ConstantPoolGen cp =
classgen.getConstantPool();
InstructionList il = new InstructionList();
StringTokenizer toks =
new StringTokenizer(expression);
int stacksize = 0;
int maxstack = 0;

while (toks.hasMoreTokens()) {
String tok = toks.nextToken();

if (tok.startsWith("$") {
int varnum = Integer
.parseInt(tok.substring(1));

// load array reference
il.append(InstructionConstants.ALOAD_1);
// load array index
il.append(new PUSH(cp, varnum));
// pull value from array
il.append(InstructionConstants.IALOAD);
} else {
int op = "+-*/".indexOf(tok.charAt(0));

switch(op) {
case -1:
int val = Integer.parseInt(tok);
il.append(new PUSH(cp, val));
break;
case 0:
il.append(InstructionConstants.IADD);
break;
case 1:
il.append(InstructionConstants.ISUB);
break;
case 2:
il.append(InstructionConstants.IMUL);
break;
case 3:
il.append(InstructionConstants.IDIV);
break;
default:
throw new RuntimeException(
"invalid operator");
}
}
}
il.append(InstructionConstants.IRETURN);

MethodGen method = new MethodGen(
Constants.ACC_PUBLIC, Type.INT,
new Type[] {Type.getType("I")},
new String[] {"args"},
"evaluate", classgen.getClassName(),
il, cp);

method.setMaxStack();
method.setMaxLocals();

classgen.addMethod(method.getMethod());
}
}


```

**sys-con  
media**

**www.sys-con.com**

Download the Code!  
www.javadevelopersjournal.com

# sitraka

[www.sitraka.com](http://www.sitraka.com)

# Under the Hood: `java.lang.reflect.Proxy`

Think more, type less



WRITTEN BY  
PAUL MCLACHLAN

This is not another article about the Proxy API as introduced in JDK 1.3. Well, okay, it is, but only indirectly. The Proxy API is implemented using techniques that can seem a lot like magic if you're not familiar with them.

That's really what I want to share: how and where to dig for more information. I'll spoil as much of the magic as I can.

*Generative programming* is a class of techniques that allows for more flexible designs without the performance overhead often encountered when following a more traditional programming style.

Individual generative techniques have been around for decades, but programming systems like Java make them much easier to build. In recent years there has been an explosion in their use.

A JSP engine is an obvious example. A generative program is simply a program that generates another program. If you've ever made a syntax error in a JSP page, you're no doubt (painfully) aware that the underlying implementation of a JSP is a servlet. You might say that the servlet was generated from your JSP on its first run.

In the implementations I've seen, it's quite easy to determine how the JSP engine works because the generated servlet code is actually written to a .java source file somewhere on disk.

On the other hand, `java.lang.reflect.Proxy` is a generative program that isn't quite as transparent – a perfect topic for the curious.

First, in case you're unfamiliar with the Proxy API, let's recap a little. Consider a class "Bug" with methods like `escalate()` and `resolve()`. Actually, pretend it has a lot more methods, or that there are a lot of classes similar to it with the same problem. Or, if you're like me, pretend that you really, really, really hate repetition. Listing 1 shows "Issue," the interface that Bug implements.

Now you have to implement a history-tracking module that can log all the actions taken on bugs. One way is to go

into every method in Bug, add `log()` calls:

```
public void resolve( Person by,
String desc ) {
    log( "Bug", "resolve",
by+", "+desc, null );
    resolved = true;
}
```

and hope you don't forget to change the "resolve" string for each call or to log a parameter or leave out a method or class entirely, and that no one adds or exposes new methods in the future. Forget being able to easily change the `log()` call to replay or rollback actions later.

You're always better off trading thinking for typing: the less typing you do, the less gunk lying around that you have to maintain. Besides, it's more fun to use an automatic Proxy. A Proxy is an automatically generated class that implements an interface that you choose. When any of the methods it implements gets called, the Proxy object will call an `InvocationHandler`'s `invoke()` method. `invoke()` can then do whatever it wishes, including logging the parameters and forwarding the call onto the "real" object. See Listing 2 for an example.

The "method.invoke()" line uses Java reflection to invoke the appropriate method on the "real" underlying Bug object and then store the return code. The rest of the method is organized so it can log even if an exception is thrown. Our `log()` will write the log entry in this format: `com.dotcom.Bug.resolve( Jim, fixed now)`.

As I mentioned earlier, this is not an article about the Proxy API, so I've deliberately left out a lot of details. For the curious, the complete example is avail-

able for download on the **JDJ** Web site ([www.sys-con.com/java/sourcec.cfm](http://www.sys-con.com/java/sourcec.cfm)).

If you've read anything about Aspect-Oriented Programming (AOP), not only will you recognize that logging is a popular example, you'll also realize that this use of the Proxy class eliminates the need to scatter logging code throughout a program. This provides something coined *separation of concerns*: keeping different aspects of the program separate. This should increase the clarity and general maintainability of the code you write because smaller pieces of it need to be grasped when looking at any particular method. When all is said and done, you've typed less. Actually, if you enjoy these concepts, you'll love AOP (look at AspectJ at [www.aspectj.org](http://www.aspectj.org)).

One thing about adding lots of `log()` calls is that by the time you finish typing, you know, in painstaking detail, what gets logged and how. It's nice to know "how things work"; debugging is easier and design decisions are, if not actually better, at least more informed.

Perhaps it's time for a little investigation into how our magic Proxy works. Unlike your options in the Microsoft world (which begin and end with SoftICE and some of the utilities at [www.sysinternals.com](http://www.sysinternals.com)), Java includes source code in the form of an `src.jar` that ships with the JDK. Once you know it's there, it doesn't take much ingenuity to extract it and pull up `src/java/lang/reflect/Proxy.java`. I use WinZip (because I'm lazy), but "jar -xvf src.jar" also works.

If you're good at squinting and looking at the big picture, you will be able to determine that Proxy builds a Java class into a byte array and then loads it into the JVM with a private native



**preemptive**  
[www.preemptive.com](http://www.preemptive.com)

defineClass0 method.

Incidentally, if you ever have the urge to do the same, `java.lang.ClassLoader` provides the API you'll need. This technique is more useful than you might initially think; Stu Halloway's excellent book, *Component Development for the Java Platform*, covers the topic extensively.

Aside from the broad approach, this source code doesn't tell us that much. For instance, we can't speculate as to how much space a Proxy object will take up or what its performance might be like. All the interesting work is being done by a `sun.misc.ProxyGenerator` class for which there is no corresponding source file, at least in `src.jar`.

If you've put in your time on Sun's Web site, you'll know that they make the JDK source available for "look but don't touch" use through the Sun Community Source License, or SCSL. As this source can be compiled into an actual JDK, we can rest assured that the code will be in there this time. If you're so inclined, take a look at `src/share/classes/sun/misc/ProxyGenerator.java`.

Sometimes your questions are most easily answered by pulling up the source code to the Sun JVM. This, sadly, is not one of those times.

When presented with the full complexity of this implementation, you'll realize (as I did) that you don't really care how Proxy works beyond the broad details. The algorithm used to build a class file byte by byte is actually uninteresting compared to the generated class itself.

If you want to learn about javac, don't study the source, create examples and study the bytecode it produces. I doubt I'll ever find it necessary to read the code to the Hotspot JIT, although I have spent a week of quality time with the x86 assembly it produces. I daresay that far fewer people care how Hotspot optimizes than what optimizations it really performs.

Unfortunately, Proxy doesn't leave a `.java` or even a `.class` file on disk for us to

examine, so we'll need a way to intercept the class bytes that it generates. An annoyingly easy way would be to modify the code to Proxy to dump the generated class bytes to disk.

Nothing prevents you from recompiling Proxy and either replacing it in `jre/lib/rt.jar` or prepending it to the JVM bootclasspath. The bootclasspath is like the classpath except it's for system classes. To prepend an entry to the bootclasspath, you need to run Java with a command line like:

```
java -Xbootclasspath/p:C:\newSysClasses
com.dotcom.LoggingTest
```

I won't go into too much detail. Ted Neward has an excellent explanation of the bootclasspath at [www.javageeks.com/papers](http://www.javageeks.com/papers). Although I think it's okay to use such hacks while exploring, there should be a more elegant way to capture class files.

If you have a tendency to read obscure parts of the API, you might be aware of a `class_load_hook` callback hidden inside the Java Virtual Machine Profiling Interface (JVMPi). JVMPi is an API used by tools vendors for profiling products such as DevPartner/Java, JProbe, or Optimizeit. It's a great way to learn about the inner goings-on of the VM and can be quite entertaining (interpret that how you will).

### The ClassCapture Tool

If you're good at JNI, writing a simple JVMPi agent that captures class bytes is quite easy, once you get over the chutzpah of it all. Here are the steps:

1. Create a DLL called "capture.dll" (or a shared object called "libcapture.so" if you're on a Unix system) that exports a `JVM_OnLoad()` function. `JVM_OnLoad()` is the function that the JVM will call when it first loads our DLL.
2. `JVM_OnLoad()` should initialize JVMPi and enable the `class_load_hook` event. We'll also pass a base directory to `JVM_OnLoad` so it knows where to write the `.class` files (see Listing 3).
3. Create a `notifyEvent()` method to listen for `class_load_hook` and write the class file bytes to disk (see Listing 4). (The omitted methods are contained in the ClassCapture source download.)
4. When you run a program, add "-Xruncapture:outputPath" to the Java command line. "-Xrun" is a prefix that tells the VM to load the named DLL - in this case "capture". The colon is used to pass arguments to the DLL;

we're using it to pass in our output directory.

You might execute:

```
java -Xruncapture:C:\proxy\captured
Classes -cp .
com.dotcom.LoggingTest
```

From then on, you'll get a call to `notifyEvent()` every time a class is loaded, regardless of where it's loaded from. Remember, classes can have many sources: normal `.class` files on disk, from a compressed `.jar` file, from a URL in Australia, or even dynamically generated, as we've discovered.

The callback gives you the actual bytes of the class (the same format found in a `.class` file on disk) and the opportunity to change them prior to their being loaded. (The ClassCapture tool [source and Win32 binary] is available for download on the *JDJ* Web site.)

Having done all this work, let's take ClassCapture for a test drive. Going back to our original example of the bug history module, the ClassCapture tool will create a directory structure like this:

```
C:\proxy\capturedClasses
+--com
  +--dotcom
    +--java
      +--io
        +--lang
          +--reflect
            +--net
              +--security
                +--cert
                  +--util
                    +--jar
                      +--zip
+--sun
  +--misc
    +--net
      +--www
        +--protocol
          +--file
            +--reflect
              +--security
                +--action
```

As you can see, even system classes are captured, except for those that were loaded before `capture.dll`, such as `java.lang.Object` or `java.lang.String`.

In `C:\proxy\capturedClasses` there's a `SProxy0.class` file. If you're fluent in Java bytecode you can disassemble it with "javap -c -classpath . capturedClasses SProxy0". If you don't habitually read about mysterious binaries in `jdk/bin`, you might not be aware that `javap` is a Java class file disassembler that ships with the JDK.

Every so often I hear of someone trying to prevent decompilation by encrypting class files on disk and decrypting them when his or her application is run. This tool should be a graphic demonstration that this approach is a waste of engineering effort.

The consensus seems to be that if you need protection for your class files, simple obfuscation can be used. If you were at JavaOne 2001 you may have heard the Hotspot team mention that more advanced code obfuscations (such as control-flow obfuscation) can actually produce slower programs: the obfuscated bytecode is more difficult to optimize.

**hit software**  
[www.hitsoftware.com](http://www.hitsoftware.com)

JVM bytecode is very simple when compared to traditional chip-based instruction sets (like x86), which makes it easy to examine and reason about, if you're into that kind of thing. Otherwise, you're still in luck because it also makes it easy to decompile. There are many (free and commercial) Java decompilers available and I haven't spent enough time with any of them to start picking favorites. A search on Google for "Java decompiler" will give you everything you need. The decompiled (okay, I air-brushed it a little) Proxy class is shown in Listing 5.

### How Proxy Works

It turns out that Proxy works as we might have guessed. It doesn't generate a .java file on disk and then compile it the way a JSP engine does, so we don't have to worry that creating lots of different Proxy objects will run multiple compiles and grind away on even the fastest machine. Instead, it efficiently builds a .class file in memory and loads it directly into the JVM.

The class it generates uses the Reflection API to look up methods, but this is done only once in the static initializer. A reflective lookup is much slower than a reflective invoke once you

have a method reference.

By looking at the `getPriority()` call you can see how it casts the object that the handler returns into integer and then calls `intValue()` to return a primitive int.

The `escalate(Person by, int amount)` method shows the boxing (new `Integer()`) that is generated in order to pass a primitive int into the handler method as an object.

The `defer()` method is a good way to see that `RuntimeExceptions`, `Errors`, and checked exceptions (such as `CannotDeferException`) are simply rethrown, but other exceptions are wrapped inside an `UndeclaredThrowableException`.

The implementations of `toString()`, `equals()`, and `hashCode()` show that the Proxy will properly call the invocation handler even for `java.lang.Object` methods.

### Conclusion

We've gone about as far as we can down this path, laid bare the internals of the Proxy API, and hopefully learned a couple of neat tricks along the way. Since you've already read this article I can admit, now, that we didn't discover anything about the Proxy API that couldn't be found in its documentation. But, having done all this work you might

find that you're no longer quite as impressed by `java.lang.Proxy`.

Proxy is a neat concept, and it does use a couple of esoteric Java APIs, but it isn't anything beyond our reach, hidden in the depths of the JVM. It isn't even that farfetched to imagine writing something similar, should the need arise. If you do so, or if you stumble across someone else who has (perhaps someone without the documentation diligence of the JDK team), maybe you'll have a couple of the pieces of the puzzle already in place when you have to try and work out what on Earth is going on. ☘

### Resources

- *Generative Programming*: [www-ia.tuilmnau.de/~czarn/generate/engl.html](http://www-ia.tuilmnau.de/~czarn/generate/engl.html)
- *Sun Community Source License*: [www.sun.com/software/communitysource/java2/index.html](http://www.sun.com/software/communitysource/java2/index.html)
- *Ted Neward's explanation of the Java BootClassPath*: [www.javageeks.com/Papers/BootClasspath/index.html](http://www.javageeks.com/Papers/BootClasspath/index.html)
- *Component Development for the Java Platform by Stuart Hallway*: <http://cseng.aw.com/book/preface/0.3829.0201753065.00.html>

pdm@acm.org

#### Listing 1: Issue interface: Issue.java

```
package com.dotcom;

public interface Issue
{
    public void escalate( Person by, int amount );
    public void resolve( Person by, String resolution );
    public void defer( Person by, String resolution )
        throws CannotDeferException;
    public int getPriority( Person by );
}
```

#### Listing 2: Generic handler method: LoggingInvocationHandler.java

```
package com.dotcom;

import java.lang.reflect.*;

class LoggingInvocationHandler
implements InvocationHandler
{
    private Class clazz;
    private Object forObject;

    public LoggingInvocationHandler( Class c, Object o )
    {
        this.clazz = c;
        this.forObject = o;
    }

    public Object invoke( Object proxy,
                        Method method,
                        Object[] args )
        throws IllegalAccessException,
        InvocationTargetException
    {
        try
        {
            // execute the underlying method:
            Object ret = method.invoke( forObject, args );

            // log the call & return code:
            LogManager.log(
                clazz.getName(),
                method.getName(),
                args,
                ret );

            return( ret );
        }
        catch( InvocationTargetException ex )
        {

```

```
// log the call & exception thrown:
LogManager.log(
    clazz.getName(),
    method.getName(),
    args,
    ex.getTargetException() );

    throw ex;
}
}
```

#### Listing 3: Class capture: JVM\_OnLoad

```
extern "C" JNIEXPORT jint JNICALL JVM_OnLoad(
    JavaVM *jvm,
    char *options,
    void *reserved )
{
    // read options into global base path variable
    parseArguments( options );

    // get a pointer to the JVMPI interface
    if( jvm->GetEnv( (void **) &jvmpci, JVMPI_VERSION_1 ) < 0 )
    {
        trace( "Error enabling JVMPI\n" );
        return JNI_ERR;
    }

    jvmpci->NotifyEvent = notifyEvent;
    jvmpci->EnableEvent( JVMPI_EVENT_CLASS_LOAD_HOOK, NULL );
    return JNI_OK;
}
```

#### Listing 4: Class capture: notifyEvent()

```
void notifyEvent( JVMPI_Event *event )
{
    if( event->event_type != JVMPI_EVENT_CLASS_LOAD_HOOK )
        return;

    //
    // We're passed an event structure like:
    //
    // struct {
    //     unsigned char *class_data;
    //     jint class_data_len;
    //     unsigned char *new_class_data;
    //     jint new_class_data_len;
    //     void * (*malloc_f)(unsigned int);
    // } class_load_hook;

    unsigned char* class_data =
```

**fiorano**  
[www.fiorano.com](http://www.fiorano.com)

```

event->u.class_load_hook.class_data;

jint class_data_len =
event->u.class_load_hook.class_data_len;

// Leave the class unchanged
copyUnchangedClassBytes( event );

// Getting the name of a class from the class
// bytes isn't exactly straightforward, I'll
// spare you the details:
const char* path = getClassPath( class_data );

// make directories if necessary
ensurePathAvailable( path );

// Write out the class bytes as we were given them
FILE* f = fopen( path, "wb" );

delete[] const_cast<char*>( path );
path = 0;

if( !f ) return;

fwrite( class_data, class_data_len, 1, f );
fclose( f );
}

```

Listing 5: Decompiled proxy class: \$Proxy0.java

```

import com.dotcom.Issue;
import com.dotcom.Person;
import java.lang.reflect.*;

public final class $Proxy0
extends Proxy implements Issue
{
public $Proxy0(InvocationHandler h)
{
super( h );
}

private static Method m_defer =
Issue.class.getMethod(
"defer",
new Class[] { Person.class, String.class }
);

private static Method m_escalate =
Issue.class.getMethod(
"escalate",
new Class[] { Person.class, Integer.TYPE }

```

```

);

private static Method m_getPriority =
Issue.class.getMethod(
"getPriority",
new Class[] { Person.class }
);

private static Method m_resolve =
Issue.class.getMethod(
"resolve",
new Class[] { Person.class, String.class }
);

private static Method m_hashCode =
Object.class.getMethod(
"hashCode",
new Class[0]
);

private static Method m_toString =
Object.class.getMethod(
"toString",
new Class[0]
);

private static Method m_equals =
Object.class.getMethod(
"equals",
new Class[] { Object.class }
);

public final void defer(Person p, String s)
throws CannotDeferException
{
try
{
h.invoke(
this,
m_defer,
new Object[] { p, s } );
}
catch( Error err ) { throw err; }
catch( RuntimeException rte ) { throw rte; }
catch( CannotDeferException cde ) { throw cde; }
catch( Throwable t )
{
throw new UndeclaredThrowableException( t );
}
}

public final void escalate(Person p, int i)

```

# purdue

[www.purdue.com](http://www.purdue.com)

# north woods

[www.northwoods.com](http://www.northwoods.com)

**new atlanta**  
[www.newatlanta.com](http://www.newatlanta.com)

```

{
    try
    {
        h.invoke(
            this,
            m_escalate,
            new Object[] { p, new Integer( i ) } );
    }
    catch( Error err ) { throw err; }
    catch( RuntimeException rte ) { throw rte; }
    catch( Throwable t )
    {
        throw new UndeclaredThrowableException( t );
    }
}

public final int hashCode()
{
    try
    {
        return (
            (Integer) h.invoke(
                this,
                m_hashCode,
                null )
            ).intValue();
    }
    catch( Error err ) { throw err; }
    catch( RuntimeException rte ) { throw rte; }
    catch( Throwable t )
    {
        throw new UndeclaredThrowableException( t );
    }
}

public final String toString()
{
    try
    {
        return (String)
            h.invoke( this, m_toString, null );
    }
    catch( Error err ) { throw err; }
    catch( RuntimeException rte ) { throw rte; }
    catch( Throwable t )
    {
        throw new UndeclaredThrowableException( t );
    }
}

public final boolean equals(Object o)
{
    try
    
```

```

{
    return (
        (Boolean) h.invoke(
            this,
            m_equals,
            new Object[] { o } )
        ).booleanValue();
    }
    catch( Error err ) { throw err; }
    catch( RuntimeException rte ) { throw rte; }
    catch( Throwable t )
    {
        throw new UndeclaredThrowableException( t );
    }
}

public final int getPriority(Person p)
{
    try
    {
        return (
            (Integer) h.invoke(
                this,
                m_getPriority,
                new Object[] { p } )
            ).intValue();
    }
    catch( Error err ) { throw err; }
    catch( RuntimeException rte ) { throw rte; }
    catch( Throwable t )
    {
        throw new UndeclaredThrowableException( t );
    }
}

public final void resolve(Person p, String s)
{
    try
    {
        h.invoke(
            this,
            m_resolve,
            new Object[] { p, s }
        );
    }
    catch( Error err ) { throw err; }
    catch( RuntimeException rte ) { throw rte; }
    catch( Throwable t )
    {
        throw new UndeclaredThrowableException( t );
    }
}
}

```



# basebeans

www.basebeans.com



# **vmgear**

[www.vmgear.com](http://www.vmgear.com)



JASON R. BRIGGS J2ME EDITOR

## Chasing the Sun

I've been hearing lately that Bluetooth is making a comeback. Considering that it had hardly gotten started when it was written off in certain quarters, it's amusing to see a comeback prediction so soon. In any case, I can see that short-range wireless protocols, such as Bluetooth, will eventually be enormously useful in the device market.

The other day I was watching someone on the Underground play a game on a Nintendo Game Boy Advance. To play a different game, he had to rummage through his pack to find the cartridge, then plug it in; by the time he'd accomplished this, he had arrived at his station. Contrast this with a Bluetooth – or similar – world, where all your devices can talk to each other (throwing away the idea of convergence for the moment), and it's a different story.

Instead of a bunch of gaming cartridges rattling around in his pack, our gamer will carry a single mobile storage device – the kind of thing I'm thinking of is a fit-in-the-palm-of-the-hand device with a series of ports capable of accepting compact flash cards, including IBM's 1GB microdrive, Dataplay's little CD thingamies, and whatever else fits in the available space. A Java Boy (because, of course, by this time Nintendo will have realized the error of their ways and replaced the Game Boy Advance with a Java device) will then be able to list all the games on the storage device (you don't even have to get it out of your bag) and transfer the selected game. Bored with the current collection, the gamer can download (and pay for, if it's commercial software) another game from the Internet, using a mobile phone/PDA that will then transmit the downloaded file to the storage facility.

All this sounds like a nice idea, and I'm sure I'm not even touching on the possibilities local wireless communications will provide, but still...I wonder when I'll actually see it.

I recently had an interesting conversation with Nazomi Communications.

Nazomi has been marketing their Java Accelerator IP for quite some time, but are now launching a Universal Java Accelerator JA108 chip for MIDP-enabled devices. Basically, the JA108 can plug into existing systems on the memory bus between the processor and SRAM/Flash; it then accelerates about 169 bytecodes (Nazomi quotes a 15x–60x acceleration for certain operations, and a 4x–6x acceleration for multimedia applications). With an extremely small form factor, either 7mm x 7mm or 10mm x 10mm, and a per unit cost of U.S.\$5.59 per unit for 10,000 units or more, it won't cost manufacturers – or more important, consumers – the world to improve Java execution performance. Java Boy, here we come?

### Goodbye, UK – Hello, New Zealand

My time in the UK is rapidly drawing to a close. As you read this missive I shall be – I hope – sitting on a train somewhere in Europe, watching snow-covered hills pass by, before moving on to warmer pastures. (Touring in winter, you ask? What can I say? My wife likes snow.)

While I will miss a number of things about this odd little island, I won't miss the weather, the smog, and the peculiar British predilection for constructing roads so narrow that only a bicyclist would feel comfortable navigating them. Hedgerows are another local tradition that escapes me. Yes, they look nice; no, I would not prefer that English country roads were without them. But combine hedgerows with narrow roads and drivers that hit blind corners at speed, pausing only to tootle their horns before entering an intersection...nope, I can't see myself missing that.

I will, however, miss being able to walk approximately two paces anywhere in London and find a pub (usually called The George). I will also miss the variety of slightly warm beers served within, and the fact that when you've imbibed too many of those beverages, you can wobble two steps back out the door and find some sort of public transport to whisk you home (okay,

### Chasing the Sun

I've been hearing lately that Bluetooth is making a comeback. Considering that it had hardly gotten started when it was written off in certain quarters, it's amusing to see a comeback prediction so soon

by Jason R. Briggs

74

### Wireless Devices – Java's Next Home

Java clarifies and simplifies an increasingly diverse wireless world

by Jeff Capone

76

### J2ME Roadmap

Here you'll find all the APIs that fall beneath J2ME's umbrella

78

### Getting Started with Java on PDAs

PDAs are becoming a permanent fixture in the everyday lives of consumers and business people.

by Rob Tiffany

80

### Boosting the Performance of Java Software on Portable Devices

Why use Java technology?

by Ron Stein

90

“whisk” is a bit of an exaggeration...all right, a lot of an exaggeration...okay, okay, I'm lying; you usually stand around waiting forever).

I'll be returning to my country of origin where, yes, it does rain, but at least it's warm; the national sport involves running around with an oval ball rather than a spherical one (for the Americans reading this, that doesn't mean wearing copious quantities of padding); and where the grass is always greener, if only because I'm predisposed to think that way.

So, goodbye, exceedingly cold winters, Tube strikes, and incredibly high living expenses; hello, fresh sweet corn that can be purchased for considerably less than the mortgage on your house, great beaches, and – cross-my-fingers – sun! ☀

▼▼▼ [jasonbriggs@sys-con.com](mailto:jasonbriggs@sys-con.com)

#### AUTHOR BIO

Jason R. Briggs is a Java analyst programmer and – sometimes – architect. He's been officially developing in Java for almost four years “unofficially for five.”

# pramati

[www.pramati.com](http://www.pramati.com)



WRITTEN BY JEFF CAPONE

# Wireless Devices — Java's Next Home

Software that leapfrogs the intelligence and usability of wireless devices is quite a captivating pitch, yet we shouldn't forget a similarly proffered claim regarding Java on PCs when it became mainstream in 1995. Ultimately, it never took off on the client side, even though it enjoys tremendous success on the server side, resulting in a dedicated and growing base of developers. What could possibly make a mobile and Java combination so much more compelling and captivating than its somewhat failed union with the PC? Will Java truly occupy the new breed of intelligent mobile devices as its new home? I believe so – and a little bit of history can help explain why.

In 1995, it was originally thought that Java's home would be client devices – the PC, Macintosh, or any other computing platform. Developers would use Java to write OS-independent programs (applets) that, when included in an HTML page, could run on client devices to enhance the user interface. Although a few Web-based applications initially incorporated applets, the client devices never became the home for Java.

Having Java on the client provided only marginal gains, and these gains did not outweigh the problems for two major reasons:

1. Because of long download times and incompatibilities among Java platforms (JVMs) and browsers, Web developers eschewed Java on the client and instead concentrated application intelligence and their coding efforts on the server side.
2. As Windows became the dominant client operating system, developers started using the native operating system interfaces, most notably the Windows APIs, to provide advanced application functions such as better user interfaces and offline operation/synchronization capabilities.

Despite client-side problems, Java successfully found a home on the server with

the introduction of Java 2 Enterprise Edition during the Internet revolution. Unlike clients, the operating systems and the hardware on servers remained heterogeneous and continued to proliferate with the introduction of Linux. Java came to the rescue by protecting the developer from operating system complexities. J2EE also rapidly displaced ad hoc server-side programming (e.g., PERL, PHP) due to its inherent advantages, including clustering/scalability, object-oriented structure, and code reuse. As a result, Java quickly proved its exceedingly low cost of ownership to the enterprise.

In the wired world, Java improved the end-user experience to a degree, but it's on thin wireless devices that Java will truly demonstrate its capabilities as it's faced with security, connectivity, and user interface issues. Unlike thick desktop clients, these wireless clients have limited screen sizes and input capabilities and immature security capabilities. In addition, they provide only intermittent connectivity to networks. Irrespective of wireless convergence and worldwide standards, hardware and operating system diversity continue to increase at a rate that perplexes even the most competent developers. But once they begin to build wireless applications using Java, developers will quickly see the exponential advantages Java offers wireless applications versus those built for the desktop.

Mobile technologies are only at the cusp of software innovation and true acceptance; the ultimate catalyst will be Java on mobile devices, officially called Java 2 Micro Edition. J2ME advantages include superior user interfaces with graphics; the ability to function offline, beyond wireless coverage; peer-to-peer networking; and improved security and consistency of applications across platforms and devices. For example, a data collection application for field workers, such as job time and expense reporting, improves significantly by offering offline operation and periodic synchronization of the data with the back-end system. A calen-

daring/scheduling application could significantly benefit from the improved UI by displaying the data in full-screen table format, a feature not currently supported by wireless phones' WAP microbrowsers. Over-the-air application downloads and updates combined with seamless data synchronization will finally unite the unique advantages of server-side and client-side application deployments for the enterprise.

The benefits are compelling, but J2ME must bear a heavy load on its shoulders if it is to replicate the success of its J2EE counterpart. To succeed, it must bridge the large gap between network connectivity and device capabilities. Moreover, as advances in hardware technologies increase the capabilities of mobile devices (such as Pocket PCs), and global economic slowdown hinders the roll-out of faster 3G wireless networks, the gap between network connectivity and device capabilities will only widen. J2ME needs to fill this gap by providing not only good user experience, but also seamless synchronization, advanced server communications, and solid peer-to-peer communications.

The wireless environment has created many new challenges that don't exist in the line-wired environment, most notably an ever-increasing gap between the capabilities of devices and the bandwidth of wireless networks. J2ME is an ideal candidate to address these unique challenges and fill the gap, making mobile devices the ideal new home for Java.

However, truly settling into its new home will require Java to enable more than just mobile gaming and entertainment applications; Java needs to enable corporate vertical market applications, where over-the-air provisioning, application updates, seamless synchronization, and advanced server and peer-to-peer communications are among the basic requirements. Java's capabilities and extensibility have never really been in doubt. It's only a matter of time until Java nicely outfits its new home and delivers on its wireless promise. ☉

jeff\_capone@aligo.com

AUTHOR BIO

Jeff Capone, Ph.D., a nationally known Java expert, has spent the past decade researching wireless applications. As Aligo's CTO, Jeff leads the technology development and is principal architect of the M-1 Server. Before Aligo, he was a professor at Arizona State University and director of the Network Engineering and Wireless Telecom Lab.

# softwired

[www.softwired.com](http://www.softwired.com)

Some of the more commonly asked questions on the various forums for J2ME seem to be, "What is J2ME?" and "Is <so-and-so-product> a part of J2ME?" Here is where you will find all the APIs that fall beneath J2ME's umbrella, and the packages you will find within those APIs.

### CONNECTED, LIMITED DEVICE CONFIGURATION (CLDC) – VERSION 1.0

java.io	input and output through data streams
java.lang	fundamental classes
java.util	collections, data and time facilities, other utilities
javax.microedition.io	generic connections classes

You can find more information on CLDC at the following URL:  
<http://java.sun.com/products/cldc/>

### CONNECTED DEVICE CONFIGURATION (CDC) – VERSION 0.2

java.io	input and output
java.lang	fundamental classes
java.lang.ref	reference object classes
java.lang.reflect	reflective information about classes
java.math	BigInteger support
java.net	networking support
java.security	security framework
java.security.cert	parsing and management of certificates
java.text	used for handling text, dates, numbers and messages
java.text.resources	contains a base class for locale elements
java.util	collections, date/time, miscellaneous functions
java.util.jar	reading Jar files
java.util.zip	reading Zip files
javax.microedition.io	connections classes

Look for more CDC information here:  
<http://java.sun.com/products/cdc/>

### MOBILE INFORMATION DEVICE PROFILE – VERSION 1.0

java.io	
java.lang	CLDC, plus an additional exception
java.util	CLDC, plus timer facilities
javax.microedition.io	networking support based upon the CLDC framework
javax.microedition.lcdui	for user interfaces for MIDP applications
javax.microedition.rms	persistent data storage
javax.microedition.midlet	defines applications and interactions between app and environment

The products page for MIDP is here:  
<http://java.sun.com/products/midp/>

### FOUNDATION PROFILE – VERSION 0.2

java.io	see CDC
java.lang	see CDC
java.lang.ref	see CDC
java.lang.reflect	see CDC
java.math	see CDC
java.net	see CDC
java.security	see CDC
java.security.cert	see CDC
java.security.acl	access control lists
java.security.interfaces	interfaces for generating keys
java.security.spec	key specifications, and algorithm parameter specifications
java.text	see CDC
java.text.resources	see CDC
java.util	see CDC
java.util.jar	see CDC
java.util.zip	see CDC
javax.microedition.io	see CDC

The profile products page is here:  
<http://java.sun.com/products/foundation/>

### J2ME RMI PROFILE (JSR #66)

This profile interoperates with J2SE RMI, and provides Java platform-to-Java platform remote method invocation for Java devices.

### J2ME GAME PROFILE (JSR #134)

This is a proposed Micro Edition specification, so nothing is yet defined. According to the JCP home page for JSR #134 (the Game Profile), the following areas will be covered:

- 3D Modeling and Rendering for Games
- 3D Physics Modeling for Games
- 3D Character Animation for Games
- 2D Rendering and Video Buffer Flipping for Games
- Game Marshalling and Networked Communication
- Streaming Media for Games
- Sound for Games
- Game Controllers
- Hardware Access for Games



PUBLISHER, PRESIDENT, AND CEO  
 FUAT A. KIRCAALI [fuat@sys-con.com](mailto:fuat@sys-con.com)  
 VICE PRESIDENT, BUSINESS DEVELOPMENT  
 GRISHA DAVIDA [grisha@sys-con.com](mailto:grisha@sys-con.com)

#### ADVERTISING

SENIOR VICE PRESIDENT, SALES AND MARKETING  
 CARMEN GONZALEZ [carmen@sys-con.com](mailto:carmen@sys-con.com)  
 VICE PRESIDENT, SALES AND MARKETING  
 MILES SILVERMAN [miles@sys-con.com](mailto:miles@sys-con.com)  
 ADVERTISING SALES DIRECTOR  
 ROBYN FORMA [robyn@sys-con.com](mailto:robyn@sys-con.com)  
 ADVERTISING ACCOUNT MANAGER  
 MEGAN RING [megan@sys-con.com](mailto:megan@sys-con.com)  
 ASSOCIATE SALES MANAGERS  
 CARRIE GEBERT [carrieg@sys-con.com](mailto:carrieg@sys-con.com)  
 KRISTIN KUHNLE [kristen@sys-con.com](mailto:kristen@sys-con.com)  
 ALISA CATALANO [alisa@sys-con.com](mailto:alisa@sys-con.com)

#### EDITORIAL

EXECUTIVE EDITOR  
 M'LOU PINKHAM [mpinkham@sys-con.com](mailto:mpinkham@sys-con.com)  
 EDITOR

NANCY VALENTINE [nancy@sys-con.com](mailto:nancy@sys-con.com)  
 MANAGING EDITOR

CHERYL VAN SISE [cheryl@sys-con.com](mailto:cheryl@sys-con.com)  
 ASSOCIATE EDITORS

JAMIE MATUSOV [jamie@sys-con.com](mailto:jamie@sys-con.com)  
 GAIL SCHULTZ [gail@sys-con.com](mailto:gail@sys-con.com)  
 JEAN CASSIDY [jean@sys-con.com](mailto:jean@sys-con.com)  
 ONLINE EDITOR

LIN GOETZ [lin@sys-con.com](mailto:lin@sys-con.com)

#### PRODUCTION

VICE PRESIDENT, PRODUCTION AND DESIGN  
 JIM MORGAN [jim@sys-con.com](mailto:jim@sys-con.com)  
 ART DIRECTOR  
 ALEX BOTERO [alex@sys-con.com](mailto:alex@sys-con.com)  
 ASSOCIATE ART DIRECTORS  
 LOUIS F. CUFFARI [louis@sys-con.com](mailto:louis@sys-con.com)  
 CATHRYN BURAK [cathyb@sys-con.com](mailto:cathyb@sys-con.com)  
 RICHARD SILVERBERG [richards@sys-con.com](mailto:richards@sys-con.com)  
 AARATHI VENKATARAMAN [aarathi@sys-con.com](mailto:aarathi@sys-con.com)

#### WEB SERVICES

WEBMASTER  
 ROBERT DIAMOND [robert@sys-con.com](mailto:robert@sys-con.com)  
 WEB DESIGNERS  
 STEPHEN KILMURRAY [stephen@sys-con.com](mailto:stephen@sys-con.com)  
 CHRISTOPHER CROCE [chris@sys-con.com](mailto:chris@sys-con.com)

#### ACCOUNTING

CHIEF FINANCIAL OFFICER  
 BRUCE KANNNER [bruce@sys-con.com](mailto:bruce@sys-con.com)  
 ASSISTANT CONTROLLER  
 JUDITH CALNAN [judith@sys-con.com](mailto:judith@sys-con.com)  
 ACCOUNTS RECEIVABLE  
 JAN BRAIDECH [jan@sys-con.com](mailto:jan@sys-con.com)  
 ACCOUNTS PAYABLE  
 JOAN LAROSE [joan@sys-con.com](mailto:joan@sys-con.com)  
 ACCOUNTING CLERK  
 BETTY WHITE [betty@sys-con.com](mailto:betty@sys-con.com)

#### SYS-CON EVENTS

VICE PRESIDENT, SYS-CON EVENTS  
 CATHY WALTERS [cathyw@sys-con.com](mailto:cathyw@sys-con.com)  
 CONFERENCE MANAGER  
 MICHAEL LYNCH [mike@sys-con.com](mailto:mike@sys-con.com)  
 SALES EXECUTIVES, EXHIBITS  
 MICHAEL PESICK [michael@sys-con.com](mailto:michael@sys-con.com)  
 RICHARD ANDERSON [richard@sys-con.com](mailto:richard@sys-con.com)  
 SHOW ASSISTANT

NIKI PANAGOPOULOS [niki@sys-con.com](mailto:niki@sys-con.com)

#### CUSTOMER RELATIONS / JDJ STORE

MANAGER, CUSTOMER RELATIONS / JDJ STORE  
 ANTHONY D. SPITZER [tony@sys-con.com](mailto:tony@sys-con.com)  
 CUSTOMER SERVICE LIAISON  
 PATTI DELVECHIO [patti@sys-con.com](mailto:patti@sys-con.com)



J2ME



J2SE



J2EE



Home

**compuware**

[www.compuware.com](http://www.compuware.com)

# Getting Started with

# on

# DA

written by Rob Tiffany

DAs are becoming a permanent fixture in the everyday lives of consumers and business people. There's no question that we have Palm to thank for bringing us a small, pen-based, easy-to-use organizer to help keep our busy lives on track.







# AS

At the same time, Microsoft has been trying to hit one out of the park for years with its Windows CE operating system running on a variety of handheld devices. As with everything else at Microsoft, it usually takes them three times to get something right, and the Pocket PC is no exception.

Based on Windows CE 3.0, the Pocket PC looks and works much like its Palm rival. However, the Pocket PC is different due to its use of more powerful hardware, more memory, and a full-blown, multithreaded operating system – it's more like a small computer than an organizer.

The success of the Pocket PC in the marketplace is largely due to the unique styling and powerful processor found in the Compaq iPAQ. With all the other Pocket PC devices shaped like unimaginative rectangular boxes, the iPAQ has had an easy time gobbling up market share. That said, the Palm is still the market-share king and isn't sitting around on its laurels waiting to become another "Netscape-style" victim of Microsoft.

As evidenced by the dominance of Windows, having the most developers writing applications for a given platform does more for its success than superior technology. The existence of a variety of programming languages for a platform plays a large role when it comes to building the critical mass of applications needed to put that platform over the top. Development for the Palm is currently being done with languages such as C/C++, CASL, NS Basic, J2ME, and AppForge (Visual Basic). Until recently, the Pocket PC development languages included eMbedded Visual Basic, NS Basic, and eMbedded Visual C++.

The Palm has clearly had the upper hand in the language department, even though its operating system lags behind the Pocket PC in features. This is quite a paradox for the Pocket PC camp, which seems to have superior technology on its side. Even the two BASIC tools for the Pocket PC are actually based on VBScript, with its lack of true data types and horrible error handling. So, the only way to take advantage of everything Windows CE 3.0 has to offer is to use an eMbedded Visual C++ tool, which most developers find intimidating at best.

Last summer, the Pocket PC gained a powerful and easy-to-use language that enabled developers take full advantage of the Windows CE operating system. Sure to make Microsoft executives cringe, Java has finally arrived for the iPAQ! Insignia Solutions ([www.insignia.com](http://www.insignia.com)) released its Jeode Embedded Virtual Machine for the Intel StrongARM processor.

The Jeode EVM is a Sun Authorized Virtual Machine that is certified and fully compliant with Sun's PersonalJava 1.2 and Embedded Java 1.0.3 specifications. It can run Java applets and applications, use a dynamic compiler to run Java apps up to six times faster than a normal JVM interpreter, and provide a preemptible, concurrent garbage collector for superior memory management. Most developers don't know much about Sun's PersonalJava and usually think about MIDP when talk of Java development for a handheld device comes up.

PersonalJava, as opposed to MIDP, wouldn't fit very well on a Palm (for example), nor could all of its classes be utilized, since the Palm OS doesn't provide all the underlying services required.

Since the iPAQ doesn't skimp on memory – 32MB is now the minimum – you don't have to compromise on your Java. PersonalJava is roughly equivalent to JDK 1.1.8 with a dash of JDK 1.2 classes and APIs thrown in for good measure. The Jeode EVM requires only 3–4.5MBs of space, depending on your need for the internationalization classes. This means that the Java Native Interface plus all of the following classes are available to you:



- **java.io:** System input and output through data streams, serialization, and the file system
- **java.util:** Collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes
- **java.util.jar:** Classes for reading and writing the JAR (Java ARchive) file format
- **java.util.zip:** Classes for reading and writing the standard Zip and GZip file formats
- **java.lang:** Core Java API classes

#### Core Java API Classes

- **java.lang.reflect:** Classes and interfaces for obtaining reflective information about classes and objects
- **java.net:** Classes for implementing networking applications
- **java.math:** Classes for performing arbitrary-precision integer arithmetic (BigInteger) and arbitrary-precision decimal arithmetic (BigDecimal)
- **java.text:** Classes and interfaces for handling text, dates, numbers, and messages
- **java.rmi:** Classes that allow Java to make remote procedure calls
- **java.rmi.dgc:** Classes and interface for RMI distributed garbage collection
- **java.rmi.registry:** A class and two interfaces for the RMI registry
- **java.rmi.server:** Classes and interfaces for supporting the server side of RMI

- **java.security:** Classes and interfaces for the security framework
- **java.security.acl:** Interface to access control list data structures
- **java.security.cert:** Classes and interfaces for parsing and managing certificates
- **java.security.interfaces:** Interfaces for generating RSA and DSA keys
- **java.security.spec:** Classes and interfaces for key specifications and algorithm parameter specifications
- **java.sql:** An API for accessing and processing data found in databases
- **java.beans:** Classes that support the creation and usage of embeddable, reusable software components
- **java.applet:** Classes necessary to create an applet and the classes an applet uses to communicate with its applet context
- **java.awt:** Classes for creating user interfaces and for painting graphics and images
- **java.awt.datatransfer:** Interfaces and classes for transferring data between and within applications
- **java.awt.event:** Interfaces and classes for dealing with different types of events fired by AWT components
- **java.awt.image:** Classes for creating and modifying images

As you can see by all the packages, classes, and APIs supported in PersonalJava, you have a handheld programming language that's more feature-rich than eMbedded Visual Basic, AppForge, NS Basic, or MIDP. Only eMbedded Visual C++ rivals PersonalJava in power, but it can't compare when it comes to developer productivity. In addition, the Jeode EVM supports the creation of console applications, and a Timer class is provided for use in animations, scheduling jobs, or any situation in which you need to execute code at set intervals.

Let's cover what PersonalJava can't do. Generally, anything that's dependent on Java 2 is off-limits, with the exception of the handful of JDK 1.2 classes found in PersonalJava. Alas, Swing is officially unsupported in this environment.

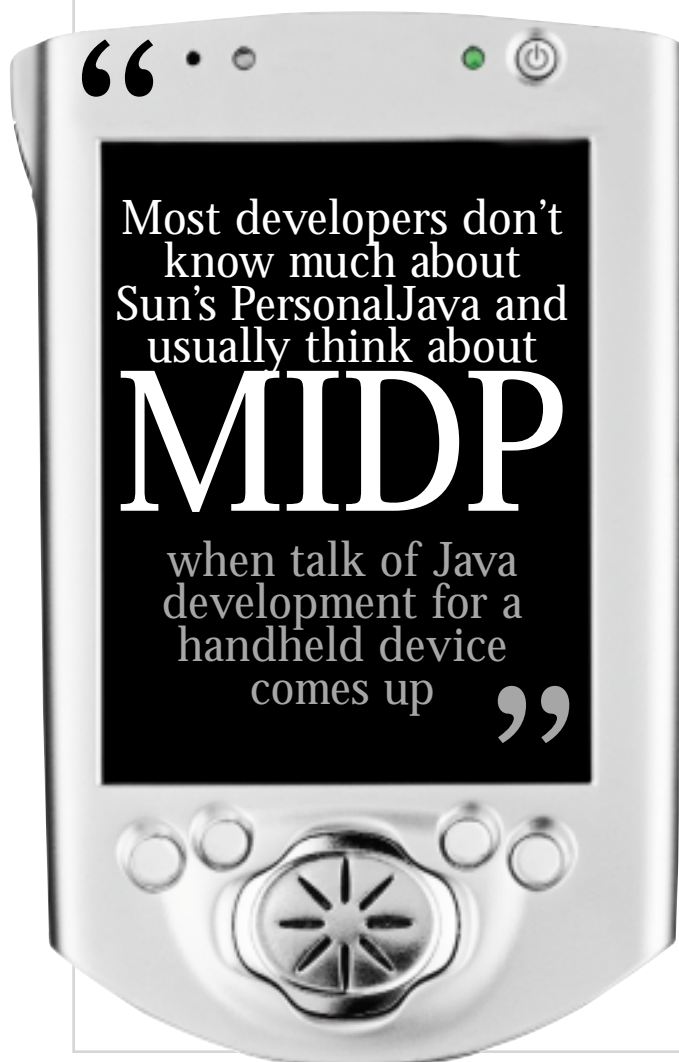
There is a workaround to make JFC 1.1.1 work with PersonalJava that requires you to change a line of code in one of the classes that make up swingall.jar. I've made this change myself and it's a good news, bad news situation. On the one hand, you get to use great-looking, Swing user interface elements on your iPAQ. The downside is the deployment of a 2MB swingall.jar file, slow program loading times, and extremely poor performance.

In my opinion, you should stick with AWT and other GUI widgets that are based on JDK 1.1. You may have to do some digging to find things like AWT-based grids and tab controls, but they're out there. My advice is to look in your old copies of "pre-Swing" JBuilder and VisualCafé to find lots of GUI JavaBeans based on AWT.

#### Building Applications

So what kind of cool applications can you build on your iPAQ with this technology?

- A single-tier application that uses JDBC to store and manipulate data on a local, pure-Java database such as Cloudscape or PointBase
- A wireless two-tier client that uses JDBC to communicate directly with enterprise databases such as Oracle and SQL Server
- A wireless *n*-tier client that uses RMI to make remote procedure calls to Java objects on a middle-tier server
- A mobile, wireless RMI server that allows other Java clients to invoke its objects



**javaone**  
[www.sun.com](http://www.sun.com)



- A chat/instant messenger client or server using Socket and URL classes
- A wireless telemetry client that remotely monitors vital corporate assets from anywhere
- A remote data-entry application to be used by personnel who work in the field.
- Miniature, mobile versions of just about any desktop or server Java application you've ever built.

I think you get the gist of where I'm going here. The best recommendation I can make is to get your hands on a wireless Ethernet (802.11b) or a CDPD (mobile Internet) card so you can start building the kind of distributed wireless applications that will take you to the next level in your career as a developer (you probably never thought that going backwards to Java 1.1 would help advance your career).

For complete information and the APIs related to PersonalJava, go to the PersonalJava application environment Web site at <http://java.sun.com/products/personaljava>.

### Getting Jeode

Now that you know what PersonalJava and the Jeode EVM are capable of, it's time to go get it. The Jeode EVM is available for purchase

on the Handango Web site at [www.handango.com/Platform/ProductDetail.jsp?siteId=1&platformId=2&productType=2&catalog=0&sectionId=0&productId=17215](http://www.handango.com/Platform/ProductDetail.jsp?siteId=1&platformId=2&productType=2&catalog=0&sectionId=0&productId=17215) for \$19.99. The software is downloadable from the Web site after you go through the shopping cart and checkout routine.

Once the software is downloaded, load it onto your Pocket PC. To begin the installation, be sure the iPAQ is in its cradle and connected to your PC via ActiveSync. Double-click on the "JeodeForArmPocketPC" executable to start the InstallShield Wizard. When asked to install Jeode in the default folder, click Yes. The installation shouldn't take more than a minute.

To verify the installation and to be sure the EVM is working properly, complete the following steps: from the Start menu on your iPAQ, navigate to Programs|Jeode|EVM to bring up the EVMConsole (see Figure 1).

The EVMConsole is equivalent to the DOS/Command Prompt you have in Windows. From here you can launch both console and AWT Java applications. There are a number of command line options that you can pass to the EVM when using the EVMConsole:

- **-?, -h, or -help:** Displays Jeode EVM help.
- **-cp <pathnames> or -classpath <pathnames>:** Here you specify the path(s) used for loading classes. Semicolons separate multiple pathnames that point to JAR files. If you need to launch a Java application that resides in a JAR file called "ipaq" that sits in the "windows" directory with a main class called "frame1", you would type in "-cp \windows\ipaq.jar frame1". To take this a step further, if your application needs classes found in a third-party JAR file, you might type in "-cp \windows\lib\jcbwt363.jar;\windows\ipaq.jar frame1". If you put your application in a package, you would type in "-cp \windows\ipaq.jar mypackage.frame1". That said, application classes that don't reside in a JAR file can be executed directly without the use of "-cp".
- **-D <propertyName> = <value>:** This supplies the value for either a Jeode EVM or Java system property. To keep the console open after an application has been executed, use the command "-Djeode.evm.console.local.keep=TRUE". If you have a console application that sends more than one screen of data to the console, use the command "-Djeode.evm.console.local.paging=TRUE".
- **-v or -verbose:** This causes messages to be displayed by the EVMConsole when a class file or dynamic library is successfully loaded or when a garbage-collection cycle is performed.
- **-version:** Displays the EVM and class library versions.
- **-Xnowinceconsole:** This option disables the EVMConsole if you don't want it to remain visible while running graphical applications.

You can further test your installation by trying out all the test applications found in the Examples and AWT folders.

I think I've done enough talking about what can be done with Java on the iPAQ, so let's put theory into practice and build some simple applications. We'll be building both a console and a GUI app that uses AWT. I'll be building Java applications that are Java 1.1-compliant, so if you're using a current IDE based on Java 2, you'll need to utilize its JDK switching feature to compile against JDK 1.1. In some cases I'll package the Java apps in JAR files - in this case, either use your IDE's JAR wizard or JAR your classes together at the command line.

### Console App

The console application we develop will demonstrate mul-

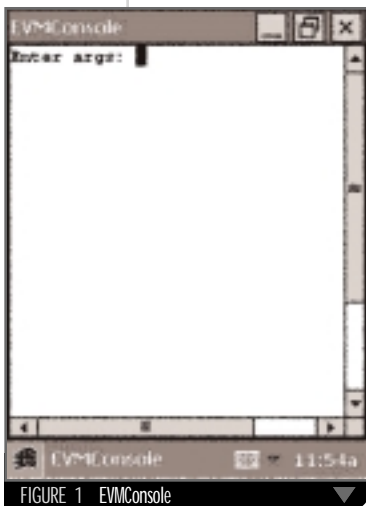


FIGURE 1 EVMConsole



FIGURE 2 My Pocket PC



FIGURE 3 Basic Console

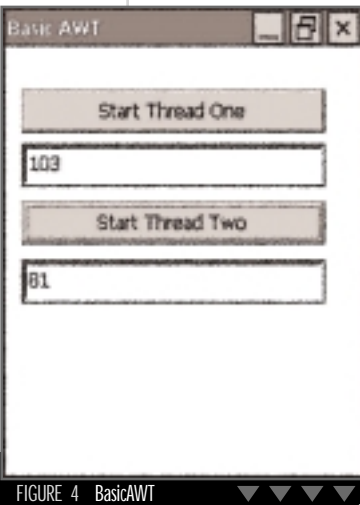


FIGURE 4 BasicAWT



FIGURE 5 Advanced GUI

**“As the Java print media market consolidates, *Java Developer’s Journal* will most likely be the only long-term player...”**

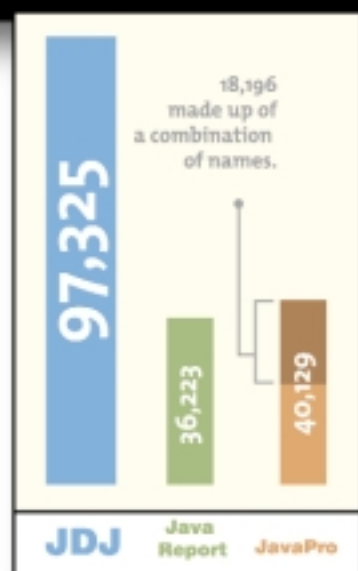
– Scott Clark, Editor of *Java Boutique/MecklerMedia*

**Now that our competitor is no longer in print...**

**BPA Web site: October 15, 2001**

Title/Topic/Home Page	Current Average Qualified Circulation	Prior History Statements
<a href="#">Java Developer’s Journal</a> Topic: COMPUTERS (Consumer) <a href="http://www.sys-con.com">http://www.sys-con.com</a>	<b>97,325</b> June-2001 <a href="#">View Report</a>	<a href="#">View History</a>
<a href="#">Java Developer’s Journal (E-mail Newsletter)</a> Topic: COMPUTERS (Business)	<b>121,462</b> November-2000 <a href="#">View Report</a>	<a href="#">View History</a>
<a href="#">Java Report</a> Topic: COMPUTERS (Consumer) <a href="http://www.javareport.com">http://www.javareport.com</a>	<b>36,223</b> June-2001 <a href="#">View Report</a>	<a href="#">View History</a>
<a href="#">JavaPro</a> Topic: COMPUTERS (Consumer) <a href="http://www.fmcsoft.com/ja">http://www.fmcsoft.com/ja</a>	<b>40,129</b> June-2001 <a href="#">View Report</a>	<a href="#">View History</a>

**Current Circulation: June 2001**



**...JDJ is here to serve you as your exclusive media partner for many years to come!**

**Historical Facts: June 2000**

	Rate Base	Actual Circulation
<b>JDJ</b>	60,000	67,618
<b>Java Report</b>	30,000	23,777
<b>JavaPro</b>	30,000	24,179

For accurate information on the circulation of Java publications, and the hands-down leadership position of *Java Developer’s Journal* and how it relates to your advertisement cost per thousand copies (CPM), please contact your account manager today.



**“Java Developer’s Journal gives us the highest number of leads on any advertising we do...”**  
Larry Humphries, President of Straka  
LIVE on SYS-CON Radio



[www.JavaDevelopersJournal.com](http://www.JavaDevelopersJournal.com)  
(201) 802-3020

tithreading and reside in a package called “basicconsole” and consist of two classes. The code for the class called “counter” is shown in Listing 1.

This class is the worker thread that simply counts upward from one number to another based on the integers that are passed into it. It sends those numbers to the console. The main class is called “app” (see Listing 2).

When this class executes, it creates two threads, passes them the number ranges, assigns thread names, then starts them.

After successfully compiling and testing these classes on your desktop computer, you need to JAR them up in a file called “BasicConsole.jar”. The next step copies this JAR file to your iPAQ.

With an open ActiveSync connection, click the ActiveSync Explore icon to view the file structure on your iPAQ. I suggest you copy your JAR file to the “My Pocket PC” folder so you won’t have to type in any path arguments to launch your application (see Figure 2).

To try out this console application on your iPAQ, bring up the Jeode console and type in “-cp BasicConsole.jar basicconsole.app”, the result of which is shown in Figure 3.

Of course, there’s not much demand for console apps anymore, so let’s build that AWT application. Keep in mind that when using your latest and greatest Java IDE, it may insert all kinds of Swing references that you’ll have to delete to create an AWT application that’s compliant with JDK 1.1.

## AWT App

The AWT application we build will also demonstrate multi-threading, reside in a package called “basicawt”, and consist of two classes. The code for the main class, called “app”, is shown in Listing 3. (*Note:* It’s rather obvious that JBuilder has been used to generate this code.)

The second class is the actual Frame that will display our GUI elements. This class is called “frame1” (see Listing 4).

The Frame loads and displays two buttons and two text boxes. Clicking on either of the buttons will spawn a thread that starts counting from one to 200. This counting will go by almost instantaneously on your desktop but will take a little more time to execute on your iPAQ.

After compiling and testing this app on your computer, JAR it up into a file called “basicawt.jar”, then copy it to the same “My Pocket PC” directory that you copied your console application to. To try out this AWT application on your iPAQ, bring up the Jeode console and type in “-cp basicawt.jar basicawt.app”. Figure 4 shows what this application should look like.

(*Note:* Clicking on either of the buttons will trigger the counting race that’s displayed in the corresponding text boxes.)

Keep in mind that this is just the tip of the iceberg when working with AWT. You can display images, check boxes, panels, combo boxes, labels, scrollbars, menus, and much more. That being said, if the AWT was so cool, why was it replaced by Swing? I can’t argue there, but I do have a suggestion. Sitraka Software (formerly KL Group) allows you to freely download their pre-Swing GUI components from [www.sitraka.com/software/jclass](http://www.sitraka.com/software/jclass). These JavaBeans make a great replacement for AWT and even rival Swing with components that include tree views, tabs, multicolumn list boxes (grids), progress meters, sliders, spinners, and windows splitters, to name a few. You can easily put together a pretty slick GUI (see Figure 5).

Now that you know what can be done with Java on the iPAQ, let’s wrap up with one more note on the deployment of your applications. You can imagine that the average user won’t be interested in launching his or her Java apps by typing commands in the Jeode console. Luckily, there’s a better way through the use of shortcuts.

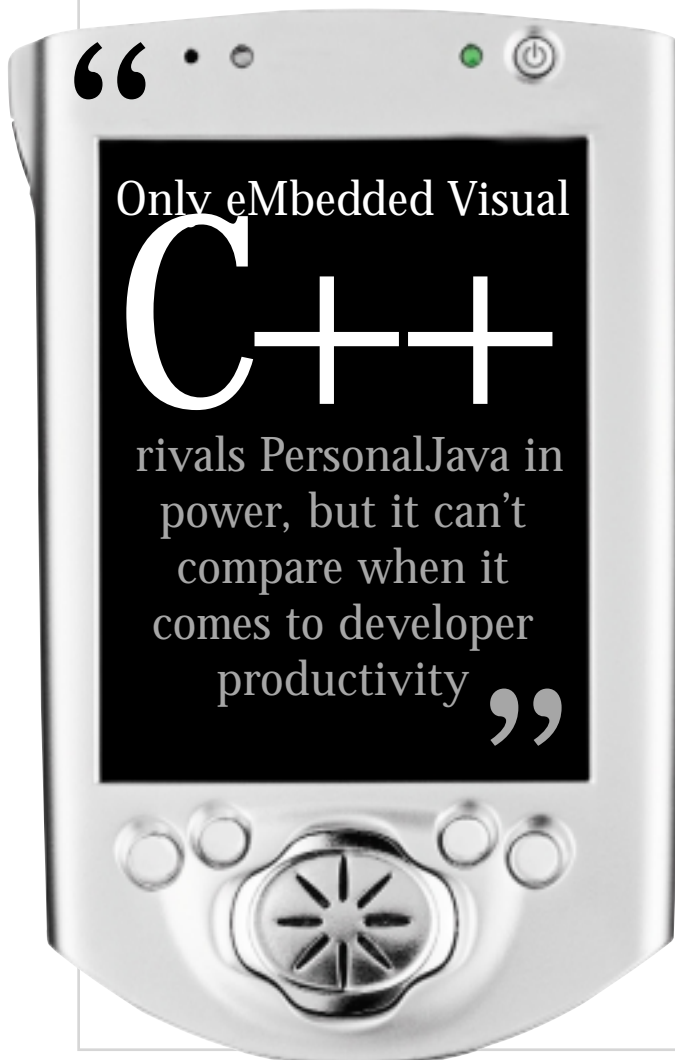
Start out by creating a text file on your desktop with a descriptive name like “BasicAWT.txt”. Now change the “txt” extension to “lnk” to turn it into a shortcut. Open this shortcut in Notepad and type in text similar to what you typed in the Jeode console. To launch your BasicAWT application, the text on the first line of your “lnk” file will look like “1#\windows\evm.exe -cp basicawt.jar basicawt.app”. All you have to do now is save and copy this file to your iPAQ in the \Windows\Start Menu directory. From now on, you’ll see the Jeode icon along with “BasicAWT” listed on the menu that you pull up from the Start button.

Congratulations! You’ve gone from learning about PersonalJava, to installing the Jeode VM and building and deploying your own Java apps. I hope you’ve enjoyed this glimpse into the world of Java on the iPAQ. Good luck in your development efforts. ☺

### AUTHOR BIO

Rob Tiffany is vice president of technology for True Quote ([truequote.com](http://truequote.com)), an online energy trading company. He has published numerous articles for a variety of magazines on topics ranging from Java servlets to wireless technologies. Rob is the author of Pocket PC Database Development with eMbedded Visual Basic from Apress.

[robtiffany@hotmail.com](mailto:robtiffany@hotmail.com)



# visual mining

[www.visualmining.com](http://www.visualmining.com)



## Listing 1

```
package basicconsole;

public class counter extends Thread {
    private int begin;
    private int end;

    public counter(int begin, int end) {
        this.begin = begin;
        this.end = end;
    }

    public void run() {
        System.out.println(this.getName() + " has begun executing ");
        for (int i = begin; i <= end; i++) {
            System.out.print(i + " ");
        }
        System.out.println("\n" + this.getName() + " has completed exe-
cuting ");
    }
}
```

## Listing 2

```
package basicconsole;

public class app {

    public app() {
    }

    public static void main(String[] args) {
        counter threadOne = new counter(1, 100);
        counter threadTwo = new counter(101, 200);
        threadOne.setName("ThreadOne");
        threadTwo.setName("ThreadTwo");
        threadOne.start();
        threadTwo.start();
    }
}
```

## Listing 3

```
package basicawt;

import java.awt.*;

public class app {
    boolean packFrame = false;

    /**Construct the application*/
    public app() {
        frame1 frame = new frame1();
        //Validate frames that have preset sizes
        //Pack frames that have useful preferred size info, e.g. from
        their layout
        if (packFrame) {
            frame.pack();
        }
        else {
            frame.validate();
        }

        //Center the window
        Dimension screenSize =
Toolkit.getDefaultToolkit().getScreenSize();
        Dimension frameSize = frame.getSize();
        if (frameSize.height > screenSize.height) {
            frameSize.height = screenSize.height;
        }
        if (frameSize.width > screenSize.width) {
            frameSize.width = screenSize.width;
        }
        frame.setLocation((screenSize.width - frameSize.width) / 2,
(screenSize.height - frameSize.height) / 2);
        frame.setVisible(true);
    }

    /**Main method*/
    public static void main(String[] args) {
        new app();
    }
}
```

## Listing 4

```
package basicawt;
```

```
import java.awt.*;
import java.awt.event.*;

public class frame1 extends Frame implements Runnable {
    Button button1 = new Button();
    TextField textField1 = new TextField();
    Thread t;
    int Count;
    Button button2 = new Button();
    TextField textField2 = new TextField();
    int textBox;

    /**Construct the frame*/
    public frame1() {
        enableEvents(AWTEvent.WINDOW_EVENT_MASK);
        try {
            jbInit();
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }

    /**Component initialization*/
    private void jbInit() throws Exception {
        button1.setLabel("Start Thread One");
        button1.setBounds(new Rectangle(14, 51, 202, 27));
        button1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(ActionEvent e) {
                button1_actionPerformed(e);
            }
        });
        this.setSize(new Dimension(235, 290));
        this.setTitle("Basic AWT");
        this.setLayout(null);
        button2.setLabel("Start Thread Two");
        button2.setBounds(new Rectangle(14, 121, 202, 27));
        button2.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(ActionEvent e) {
                button2_actionPerformed(e);
            }
        });
        textField2.setBounds(new Rectangle(14, 158, 202, 27));
        textField1.setBounds(new Rectangle(14, 85, 202, 27));
        this.add(textField2, null);
        this.add(button1, null);
        this.add(textField1, null);
        this.add(button2, null);
    }

    /**Overridden so we can exit when window is closed*/
    protected void processWindowEvent(WindowEvent e) {
        super.processWindowEvent(e);
        if (e.getID() == WindowEvent.WINDOW_CLOSING) {
            System.exit(0);
        }
    }

    void button1_actionPerformed(ActionEvent e) {
        Count = 0;
        textBox = 1;
        t = new Thread(this);
        t.setName("ThreadOne");
        t.start();
    }

    public void run() {
        if (textBox == 1) {
            for (int i = 0; i <= 200; i++) {
                textField1.setText(i + " ");
            }
        }
        else {
            for (int i = 0; i <= 200; i++) {
                textField2.setText(i + " ");
            }
        }
    }

    void button2_actionPerformed(ActionEvent e) {
        Count = 0;
        textBox = 2;
        t = new Thread(this);
        t.setName("ThreadTwo");
        t.start();
    }
}
```



## Flashline/TogetherSoft to Improve Software Development Life Cycle (Cleveland / Raleigh, NC) –

Flashline has integrated its enterprise reuse solution, Flashline Component Manager, Enterprise Edition 3 (CMEE), with TogetherSoft's Together ControlCenter 5.5 (TCC). The combined solution provides a comprehensive environment to model, build, deploy, and reuse quality software assets throughout the enterprise, resulting in increased developer productivity and measurable ROI.

[www.flashline.com](http://www.flashline.com)  
[www.togethersoft.com](http://www.togethersoft.com)



## Oracle Unveils Oracle9i JDeveloper (Redwood Shores, CA) –

Oracle is shipping a new Java development productivity kit that lets Java developers customize J2EE, XML, and Microsoft SQL Server in one IDE. JDeveloper also has an Add-In Kit that links the system with third-party software and open source tools.

The new application is available for free download through Oracle's Technology Network.

[www.oracle.com](http://www.oracle.com)



## SpiritSoft Announces Universal Caching Framework (Boston) –

SpiritSoft released SpiritCache 1.2, a standards-based caching framework that increases performance of distributed systems from wired to wireless. It supports complex, multi-tiered applications across diverse platforms and devices.



SpiritCache 1.2 features last image and delta caching of rapidly changing information that's being sent around the network, multilevel caching services, and an event notification mechanism.

[www.spirit-soft.com](http://www.spirit-soft.com)

## Slangsoft Announces MIDP Release of iTID Platform (Boston) –

Slangsoft's intelligent Text Input and Display Platform (iTID) is now compatible with the set of Java APIs that constitute the Mobile Information

Device Profile (MIDP).

The iTID platform supports complex states of input, such as the simultaneous input from phone key-

pads and voice commands.

[www.slangsoft.com](http://www.slangsoft.com)

Zeosoft Announces Java-Based Operating System Partnership (Scottsdale, AZ) – Zeosoft, a businessware company bringing networked mobile servers to business, integrated its ZeoSphere embedded application server with the SavaJe XE Java-based operating system, resulting in an improvement in performance over similar configurations on non-Java operating systems.

ZeoSphere is currently in beta



with plans for a general availability market launch in Q1 2002.

[www.zeosoft.com](http://www.zeosoft.com)  
[www.savaje.com](http://www.savaje.com)



## Pramati Server 3.0 Passes J2EE Compatibility Test Suite (San Jose, CA) –

Pramati Technologies introduces Pramati Server 3.0, the first independently developed application server to achieve J2EE 1.3 compatibility.



Pramati Server 3.0 carries a high-performance EJB 2.0 container with support for enterprise-level features such as load balancing, failover, and hot deployment.

[www.pramati.com](http://www.pramati.com)

## Pingtel Announces New Software for xpressa (Woburn, MA) –

Pingtel announced Release 1.2 software for Pingtel xpressa, a Java-based, voice-over-IP phone.



Release 1.2 offers enterprises and service providers enhanced functionality, deeper support for the SIP industry standard, and additional features not possible with conventional business phones. The new software, which is now available as a free upgrade from Pingtel's Web site, also offers a richer platform for third-party Java developers who are creat-

ing new applications.  
[www.pingtel.com](http://www.pingtel.com)

Sybase Announces PowerJ 4.0 (Emeryville, CA) – Sybase, Inc., announced the availability of PowerJ 4.0, an IDE supporting complete application and component-level development, deployment, and monitoring/maintenance for the J2EE platform. PowerJ supports Sybase EAServer and solutions powered by



it including Sybase EP. PowerJ 4.0 offers a new application server view and application server backwards compatibility. Additional features include JPDA debugging, EJB 2.0 and J2EE v.1.3 support, and new XML-based workspaces.

[www.sybase.com](http://www.sybase.com)

## IONA/TTM to Provide Integration Solutions for Legacy BEA Products (Waltham, MA) –

IONA and Total Transaction Management (TTM) announced the availability and certification of CtO-JBean for IONA's Orbix E2A Application Server Platform.

CtO-JBean is a J2EE-compliant adapter that allows seamless and scalable access to BEA's middleware products (Tuxedo and TOP END) from IONA's Orbix E2A Application Server Platform.

[www.tmsolutions.com](http://www.tmsolutions.com)  
[www.iona.com](http://www.iona.com)



## Vettanna Offers Career Path Training (San Rafael, CA) –

Vettanna, LLC, a global provider of software testing and Web services, now offers training in the career of software testing. Career Path Training offers insight to methodology and applicable tools for testers at every stage in a career.

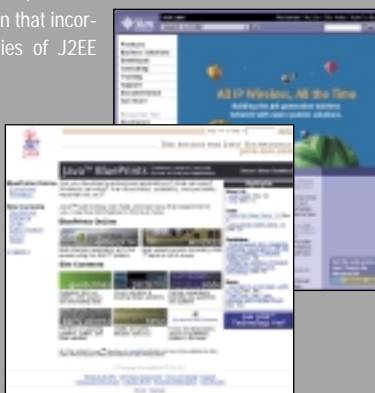


[www.vettanna.com](http://www.vettanna.com)

## SUN ANNOUNCES LATEST VERSION OF JAVA BLUEPRINTS

(Santa Clara, CA) – Sun Microsystems, Inc., announced the availability of its latest version of Java BluePrints for the Enterprise program. The program includes the Java Pet Store 1.3, the first BluePrints sample application that incorporates the new technologies of J2EE v1.3.

The Java BluePrints for Enterprise program is available at <http://java.sun.com/blueprints>, and is distributed with the Java Jumpstart CD, the J2EE Developer Resource Kit CD, and is also available from J2EE licensees who are planning to redistribute it with their products.



# Boosting the Performance of Java Software on Portable Devices



WRITTEN BY  
RON STEIN

The so-called duopoly of Intel and Microsoft brought one great advantage to personal computing – a uniform runtime platform for application software. Application software developers need only code the application using the Win32 API and compile for Pentium to be assured that their programs will run on a majority of desktops.

In contrast, Internet appliance devices are nonhomogeneous, using many different types and variants of microprocessors and many different operating systems. What this means is that software developed for one device will most likely not run on another device unchanged because of OS and/or binary incompatibilities. (Note: For the purposes of this article, a platform means the combination of a microprocessor and an operating system that includes event management and an API that supports a consistent GUI.)

You should also consider the ubiquitous expansion of the Internet beyond the realm of PCs. No longer are PCs (including Apple and Unix systems) the sole conduit for Internet access. Portable devices that include Web tablets, PDAs, and wireless handsets are increasingly enabled with Internet-access capability. Following the trend established with PCs, content delivered via the Internet will be a combination of static information, dynamic personalized information, and interactive content that includes applications.

Consider the case of Short Messaging Services (SMS) on wireless handsets.

Each wireless services operator has a different protocol and capability for piggybacking SMS over its existing digital network. This requires a carefully developed SMS client integrated into any wireless handset that matches the protocol and capabilities of the carrier. Imagine the value to a wireless services operator if they had to develop only one SMS client program that could be downloaded into any wireless handset. Further, imagine that the SMS client program could be upgraded as the capabilities of its network expands, enabling subscribers to readily take advantage of the expanded services.

As interactivity and application software become an ever more important aspect of Internet connectivity and services, the necessity for a universal runtime platform becomes quite apparent. However, unless the runtime platform is based on exact standard components, as in the case of “Wintel” PCs, a resource, cost, and performance impact accompanies the inclusion of a universal intermediate runtime platform. Therefore, an ideal universal runtime environment must possess the following properties:

1. *The platform must offer binary compatibility.* Executable software is delivered as “binary object code” or “binaries.” For example, software developed for a Macintosh computer is compiled to PowerPC binary object code, whereas software developed for an IBM-compatible PC is compiled to Pentium binary object code. As the PowerPC and Pentium microprocessors have different architectures and instruction sets, binaries for one are not interchangeable with the other. Hence the creation of an abstract runtime environment is needed to pro-

## Why use Java technology?

vide a bridge between an underlying microprocessor and application software binaries.

2. *The platform must include a microprocessor, or an emulation of one,* that includes a universal machine code that’s fully separated from the underlying microprocessor’s machine code.
3. *The platform must include an operating system with an API and corresponding capabilities that support a consistent GUI.* An API is a set of software functions that perform typical operations such as opening files, reading/writing data, allocating and managing memory, handling events, and displaying text and graphics. For application software to be truly portable, a device must possess a common set of capabilities as well as extend those capabilities to software developers via a consistent API.
4. *The platform must not impose heavy resource requirements on a system.* Portable devices are price-sensitive and must meet specific price points. Therefore it’s necessary that the runtime environment doesn’t cause a significant increase in the bill of materials and hence the overall cost of the device. Also, the leaner the requirements necessary to support a runtime environment, the broader the range of devices that can be supported.
5. *The platform must be energy efficient,* which is an absolute necessity for battery-powered portable devices.
6. *The platform must be secure,* while being Internet-aware.

Java with its “write once, run anywhere” promise possesses such properties and was designed by Sun Microsystems to address the related

Industry analysts predict that the information appliance segment will grow considerably faster than the PC market. International Data Corp. projects that by 2004, the segment worldwide will be valued at \$17.8 billion, with an installed base of 89 million units.

requirements. However, for embedded applications such as portable devices, Java has on the whole been slow to catch on until now, primarily due to performance issues and the costly system resources required for integrating Java technology.

Wireless telephone handsets are now incorporating Java technology to accept Java applications through a wireless network and run them on the mobile telephone. PDA manufacturers are beginning to utilize the Java platform as their standard and sole operating environment, allowing the manufacturer to deliver a highly differentiated and extensible product to market. Television set-top unit manufacturers have also embraced Java technology as a means to enable the delivery of interactive services (such as electronic program guides) and application software, as well as to



FIGURE 1 The Java Runtime Environment

add new functionality to their devices over time.

The value of a universal runtime platform, such as Java's, is clear, and the trend to incorporate such platforms into portable Internet-enabled devices is also clear. However, two primary issues remain before the floodgates open: the performance with which the system executes Java software and the cost to include the platform.

### The Java Runtime Environment

The Java Runtime Environment (JRE) has been designed to provide runtime environment capabilities similar to, for example, a PC, an Apple Mac, a PocketPC device, or a PalmPilot. The Java 2 Micro Edition (J2ME) is the specific JRE for portable devices, Internet appliances, and other miscellaneous embedded applications. Figure 1 shows the functional blocks of a JRE that's

made up of two primary components – a Java Virtual Machine (JVM) and a specific set of configuration and profile class libraries

The primary components of the JVM that impact performance are:

1. **Java bytecode interpreter:** Interprets Java bytecode instructions into native microprocessor instructions and is the heart of the JVM, and consequently the main performance bottleneck. The execution of bytecodes is where the biggest performance bottleneck exists, because each bytecode must be interpreted one-by-one into the native instructions of the native microprocessor.

Each Java bytecode instruction typically interprets to several native microprocessor instructions. It should be noted that the JVM and its corresponding instruction set are based on a stack architecture. The implications of a stack-based architecture will be discussed later.

2. **Class manager:** Determines which class files to load and when. The class loader also scans class files and individual bytecodes for malicious code (such as viruses) as part of the intrinsic security of a JVM. Once a class file has been loaded and verified, its bytecodes are passed on to the interpreter for execution.

3. **Garbage collector:** Reclaims memory that's no longer used. Because the Java programming language is object-oriented with an integrated garbage collection mechanism, programmers are freed from directly managing memory resources. Instead, objects are created and use memory on the fly as they're instantiated, and the memory that they use is released when the garbage collection process determines that the object is no longer needed. The garbage collection process runs concurrently with any Java software that's running, and therefore directly impacts runtime performance.

### Java Software Performance Issues

Programs developed in the high-level Java programming language are compiled to the universal binary intermediate language referred to as Java bytecode instructions. When Java software is executed, the Java bytecode instructions are interpreted to the native instruction set of the microprocessor on which the JRE has been ported and runs. This makes the bytecodes an intermediate language and the JRE an intermedi-

ate runtime platform.

An intermediate runtime platform such as the J2ME typically suffers from a performance disadvantage (as compared to natively compiled software), because the underlying system consisting of the microprocessor plus operating system must simultaneously run the intermediate platform as well as the application software. Put another way, Java software execution is handicapped because a system must execute two programs simultaneously to run a Java program – the JVM and the Java program itself. In addition to more heavily utilizing the CPU to execute Java software, the Java platform requires additional memory for its footprint and runtime needs. This gives rise to the clear goals of improving Java software execution performance and minimizing memory requirements.

Before talking about increasing Java software performance, it's helpful to clearly understand the pertinent performance issues:

1. As previously mentioned, Java software binaries are executed by a JVM that interprets each Java bytecode instruction into native microprocessor instructions – a system must run a JVM program that in turn runs Java software. In other words, to run a Java program the system concurrently executes two programs and two different instruction streams.
2. Typical commercial microprocessors, especially those used in portable devices, have register-based architectures; however, the JVM is stack-based. This is important because executing Java bytecode instructions differs significantly from the way commercial microprocessors operate. In particular, temporary data, values, and method arguments are passed through variables and a common stack.

The Java stack resides in the system's memory, which contributes to performance challenges because each stack interaction requires a memory transaction. Significant performance gains can be achieved by the careful localization of variables and stack entries within the CPU, which in effect helps to bridge the gap

The Java platform provides a universal runtime environment upon which Java software can be run.



between the stack-based JVM and register-based microprocessors.

### Boosting the Performance of Java Software

Portable devices are at a disadvantage when compared to desktop and server computers because they can't be made to take advantage of higher processor speeds, larger memories, and persistent storage available with hard disk drives. So the techniques to speed-up Java software execution on desktop and server computers are wholly inappropriate for portable, cost-sensitive consumer devices.

Performance-boosting techniques for portable devices fall into four categories:

1. Improve the performance of the underlying system. The typical option is to increase the speed of the microprocessor. This approach does increase performance but only moderately. In most systems the microprocessor runs 2-10x faster than the system's memory, so increasing the clock frequency only causes additional waiting for memory accesses such as interactions with the Java stack. Thus there is not a linear relationship between Java software performance and the clock frequency of the system's microprocessor. An additional detriment is that it raises the system cost and consumes more power. Because this is such a poor option, particularly on cost- and energy-conscious devices, this technique won't be discussed further.
2. While not specifically a compilation technique, another common method to boost performance is to carefully optimize and tune the JVM software to take full advantage of the target hardware. This often involves coding the bytecode interpretation loop in the assembler and utilizing CPU registers for common data. Other common optimizations include streamlining method invocations and optimizing the garbage collection process. Only one drawback exists to this approach: the resultant JRE becomes quite dependent on and tied to a specific hardware configuration.
3. Compilation – the ultimate goal is for

“For embedded device designs where cost, system resources, and power consumption are critical, hardware-based acceleration is rapidly becoming the preferred choice”

Java software performance to be equivalent to software compiled directly to the microprocessor's native machine code. Several techniques, described later, that compile Java software to native machine code are used to boost performance. Compilation techniques do result in performance gains, and often shine in static benchmark tests. However, compilation techniques are contrary to some aspects of the Java-platform paradigm, can't address energy conservation issues, increase memory requirements, and are limited in their capabilities.

4. As is often the case, hardware-based acceleration delivers greater performance, energy efficiency, and cost-effectiveness than can be achieved with software.

### To Compile or Not to Compile

That is indeed the question. The usual way to enable a device with Java technology is to acquire the J2ME development kit from Sun Microsystems and port the JVM, configuration, and appropriate profile(s) to a specific target system. Up until now, the most common way to boost performance was to use a JVM augmented with compilation technology (discussed in detail later). Using a compilation-augmented JVM is also a relatively fast and easy way to introduce performance-boosting capabilities into a Java-enabled device. While it's tempting to think it's also the most cost-effective method, be aware of the memory requirements.

Regardless of the compilation method used, a common set of drawbacks accompanies compilation, such as greatly expanded memory usage and inconsistent execution performance. Compilation can be done several ways.

### Ahead-of-Time (AOT) Compilation

This is where the Java program is compiled in totality prior to being executed.

Two methods are used for AOT compilation. "Way-ahead-of-time compilation" is when the program is compiled, saved, and distributed as a native executable, much in the same way that a C or C++ program is compiled, saved, and distributed. This option converts the Java program to a system-specific program and hence it's no longer portable.

The other method is when the program is distributed as a standard Java class file and compiled when it's loaded but before being executed. This latter option causes a (sometimes lengthy) delay before the program can begin running, which is typically unacceptable for transient software (such as applets and MIDlets).

### Real-Time Compilation (Just-in-Time Compilation)

This is the most common technique used in portable consumer devices where Java applications are likely to be transient. This is where a compiler is integrated into the JVM to compile portions of a Java program concurrently while the JVM is running a Java program. Two primary JIT compilation methods are commonly used. JIT compilation is where individual methods and/or classes that make up the program are compiled when loaded but before being executed. The other method is when an executing program is analyzed to determine frequently used classes, methods, and/or code fragments. These code sections are thereafter compiled and executed as native code. This latter case is referred to as adaptive compilation, statistical JIT compilation, or incremental JIT compilation.

Both options expand memory use and result in inconsistent performance because of the delays that occur during the compilation process. Furthermore, the dynamic compilation option typically uses only a relatively small amount of memory to store compiled code segments.

The combination of an interpreted intermediate language and a stack-based architecture contribute to the slow execution of Java software.



ments. As these segments change, the performance resulting from dynamic compilation is the most inconsistent because at different times a section of code may execute either slowly or quickly.

Table 1 lists the pros and cons of the different acceleration solutions.

### Hardware Is Happening

Hardware-based acceleration of typical software functions (traditional 3D rendering now being performed by 3D graphics hardware is a good example of this) typically delivers the highest possible performance due to the speed and efficiency at which dedicated hardware can execute. Hardware-based acceleration of Java bytecode execution is no exception to this rule. You should be able to intuitively appreciate that dedicated hardware can offer greater performance than software that must share CPU resources with a running JVM. Hardware Java acceleration solution vendors claim average application performance improvements from 5-15 times over that of a

standard JVM, as well as varying energy efficiencies (versus a Sun Microsystems reference implementation as the baseline). Of course, performance is also impacted by other components within a system, such as the speed of the memory and the capabilities of the host application microprocessor (if applicable).

Another noteworthy item is that some hardware solutions are transparent to the operation of a system, with the clear exception of the JVM. Such solutions tend to be design-friendly because they can be readily integrated into a system without changing or affecting the bios, operating system, or other legacy software, while also leveraging existing hardware designs, development tools, and in-house expertise.

It should be pointed out that a Java accelerator complements a JVM but neither replaces nor substitutes it. A Java Virtual Machine has several components, only one of which – the Java bytecode instruction interpreter loop – is enhanced or replaced by most accelerator solutions. The JVM still takes care

of loading and verifying Java classes, managing memory, scheduling tasks, and performing other housekeeping functions. The more efficiently these other functions are implemented, the greater the improvement a Java bytecode execution accelerator can contribute to overall system performance.

For embedded device designs where cost, system resources, and power consumption are critical, hardware-based acceleration is rapidly becoming the preferred choice. Consider that in today's systems the difference between the speed at which microprocessors and memory are clocked can be as much as 10 to 1. This too will be the case as microprocessors are run faster in portable devices and Internet appliances.

Software-based acceleration using compilation is then handicapped by the speed of system-memory devices, whereas hardware-based acceleration can include technology to compensate for slow memory. A recent study from Penn State University, published by USENIX, confirms that even though the accelerator device draws power, hard-

	INCREASING CPU CLOCK FREQUENCY	OPTIMIZING AND TUNING	AOT COMPILATION	JIT COMPILATION	HARDWARE
System Transparency	No	Yes	Yes	Yes	Varies
Device/System Integration Effort	High None None None	Moderate to High			
JVM Adaptation Effort	None	High	High	High	Moderate
Device/System Time To Market	Moderate	Fast	Fast	Fast	Moderate
Memory Usage and Requirements (excluding hardware JIT compilers)	N/A	Good	Poor	Poor to Moderate	None
Overall Cost	High	Low	High	Moderate	Low
Java Software Speed	Poor	Good	Best	Good	Best
Java Software Execution Consistency	Best	Best	Best	Erratic	Best
Energy Efficiency	Poor	Good	Good	Good	Best

TABLE 1 Pros and cons of the different acceleration solutions.

	MICROPROCESSOR EXTENSIONS	JAVA ACCELERATORS	NATIVE JAVA MICROPROCESSORS
System Transparency	Yes (if no ISA changes)	Yes	No
Device/System Integration Effort	Low	Low	High <sup>3</sup>
JVM Adaptation Effort	Low	Low	High – the JVM must be custom coded <sup>3</sup>
Device/System Time To Market	Varies <sup>1</sup>	Fast <sup>2</sup>	Slow – due to custom system design <sup>3</sup>
Java Software Performance	Good	Good	Best
Energy Efficiency	Best	Neutral to Moderate	Good

TABLE 2 Pros and cons of different hardware acceleration solutions

1. Microprocessor extensions come about by integrating Java acceleration technology with a microprocessor's core. As the supply of Java-enhanced microprocessors proliferates into the market, the time-to-market will be favorably impacted. Absent the broad availability of Java-enhanced microprocessors, this technology is mostly limited to vertically integrated companies and partnerships.
2. Existing or new devices can readily be enhanced or designed, respectively, to accommodate a Java accelerator chip.
3. An entire system design that includes all system software must be developed to accommodate an NJM.

ware-assisted Java software execution is the most energy-efficient choice, something that's especially important in battery-powered devices.

Further validating the value proposition of hardware-based, Java performance-boosting solutions are leading market analysts, such as Cahners In-Stat, that predict that within the next few years the majority of JVMs integrated into portable devices, Internet appliances, and other miscellaneous embedded applications will utilize hardware-assisted acceleration.

#### AUTHOR BIO

Ron Stein is a senior marketing manager at Nazomi Communications, Inc. He has more than 20 years' experience with Java and embedded software. Prior to Nazomi, he managed product marketing for Insignia Solutions. Ron holds an MBA from Santa Clara University and a BSEE from the University of Pittsburgh.

Hardware solutions can be segmented into three categories:

1. Microprocessor extensions include two implementation methods – instruction path interpreters and instruction set extensions. Both methods enable a microprocessor to interpret Java bytecode instructions as part of their normal processing capabilities. Such Java interpretation extensions are analogous to the way the MMX extensions in a Pentium microprocessor enable accelerated processing of multimedia in addition to standard x86 instructions.
2. Java accelerators are standalone chips or components within a System-on-Chip (SoC) that directly execute Java bytecode instructions. One specific type

of technology or chip in this category is a hardware JIT compiler that, of course, shares the same drawbacks as software JIT compilers. Excluding hardware JIT compilers, Java acceleration chips are analogous to the way graphics accelerators complement a microprocessor by performing many complex pixel manipulations and rendering functions directly with the video memory.

With regard to energy efficiency, a dedicated Java bytecode execution engine is most likely a less complex chip and so has fewer transistors than a larger general-purpose microprocessor, drawing less power and executing bytecodes faster than a microprocessor can interpret. Thus such a chip can contribute energy efficiency to a system's attributes. Such energy efficiency doesn't apply to either software or hardware JIT compilers. In other words, dedicated hardware is typically more efficient at performing specific tasks than a general-purpose microprocessor.

3. Native Java microprocessors (NJMs) are unique microprocessors that use Java bytecode instructions as the native instruction set. Some NJMs may have additional instructions that are outside the scope of executing Java software to support OS and driver development.

It's tempting to think of an NJM as the most natural choice for accelerating Java bytecode instruction execution, hence boosting the performance of Java software. However, such devices are contrary to typical device manufacturers' desires to leverage in-house expertise and assets, as well as use cost-effective standard components and technologies. The device manufacturer is prevented from acquiring and using readily available microprocessors, I/O devices and corresponding drivers, operating system software, and development tools. Rather the device designer and manufacturer are faced with the unpalatable requirement to acquire new and specific items, if available; if not available, those items must be developed from scratch. For these reasons NJMs have consistently failed to achieve broad acceptance, even NJMs

from Sun Microsystems, the pioneer of the Java platform.

Table 2 lists the pros and cons of the different hardware acceleration solutions.

#### Summary

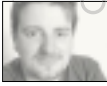
While many options exist for boosting the performance of Java software executing on Internet appliances, portable devices, and the like, the appropriate selection is neither simple nor obvious. The choice of a particular technology, option, or product should be driven by a careful assessment of market and device requirements. It should also be clear that among the solutions presented, hardware solutions offer the best balance between cost, complexity, design-friendliness, performance, and energy efficiency. It shouldn't be a surprise that hardware solutions, particularly microprocessor extensions and dedicated accelerator chips, are becoming standard components. Furthermore, system transparency and design-friendliness are critical attributes that are often overlooked in hardware solutions.

No discussion about software performance is complete without discussing benchmarks. In this regard, it should be noted that scores resulting from real-time compilation are typically not representative of how actual Java application software will perform. In fact, compiled code that exhibits benchmark scores 10–20 times faster than interpreted code may result in execution that is only one to two times faster for real-world Java application software. This is particularly true of dynamic or statistical JIT compilers, because most benchmark tests are designed as loops. JIT compilers compile and execute the loops very fast, skewing the scores (for example, a loop that runs for 1,000 iterations may run interpreted and slowly for five iterations, then compiled and fast for the remaining 995 iterations).

Hardware-based acceleration, unlike software-based acceleration, generally delivers a very consistent benchmark and real-world application performance. In other words, if a particular hardware device or technology delivers a tenfold increase in benchmark scores, it's reasonable to assume that real-world Java applications will execute 10 times faster. Be aware that Java benchmarks are heavily impacted by the system on which the JRE is running, so normalized comparisons are sometimes irrelevant if not meaningless. Other failings of benchmarks are that they don't characterize system resource utilization (such as memory) or energy efficiency. ☛

# aqua nius

[www.aquanius.com](http://www.aquanius.com)



—continued from page 5

LooksWeird...  
SoundsGreat!

When asked to define "great" he said, "I want to write stuff that the whole world will read, stuff that people will react to on a truly emotional level, stuff that will make them scream, cry, howl in pain and anger!" He now works for Microsoft, writing error messages.

The irony is that it's closer to the truth than you may want to believe. When you realize that the majority of the planet has a Microsoft product somewhere in their daily lives, that's a more powerful channel than any news mogul could ever hope to achieve.

Speaking of breaking out of the Microsoft habit, I'd like thank all those who e-mailed me the URL to JEdit ([www.jedit.org](http://www.jedit.org)), the free Java editor. Normally I'm not a big fan of Java GUIs as they're so damn slow compared to their native counterparts. I don't care what anyone says, Swing is definitely nowhere near ready for mainstream use. It's the black sheep of the Java family that needs some serious attention. (I'll go into more depth on this

subject next month.) As you know I've abandoned the bloated IDE world in favor of a back-to-basics approach to development. This has been a painless move, and I have to say the whole team here at n-ary has not uttered a single objection, especially now that we have JEdit.

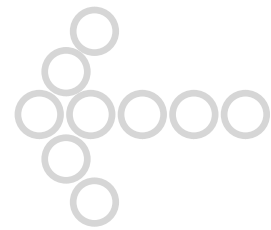
Wow. This is how software should work. Not only is JEdit feature-rich, but for those of you who still want to have your cake and eat it too, just download a simple plug-in from the masses that are available and restart. And you know the easy bit? It's all done from within the editor itself. No bugging around with URLs and complicated README files. It's an absolute joy.

Ever since I published my move to ANT, I've received a flood of e-mails supporting the move and describing how they've gone through the same process to find success. I even had an e-mail from an unnamed source working for one of the big IDE vendors who didn't even use their IDE to code their next release. Mmmm.

If I were an IDE vendor I would be seriously looking at this trend to see if it's just a blip or a serious social shift. Interestingly enough, this month I learned that there's a product that operates along the same lines as ANT ([www.openmake.com](http://www.openmake.com)). So keep an eye on JDJ as we check this out in the next couple of months. Personally, I think there's an underground movement by computer programmers to prove themselves to be real developers, since each new IDE release seems to remove you further from the actual task of producing code. If this is the case, then I can sympathize.

I'd love to hear what you think. You can find me (plus a lot of other JDJ bods) lurking around in one of the best mailing lists around, [http://groups.yahoo.com/group/straight\\_talking\\_java/](http://groups.yahoo.com/group/straight_talking_java/).

I'm off to prepare for my pre-pre-JavaOne briefings! ☺



# Coming Next Month

**JAVA DEVELOPERS JOURNAL** & **TogetherSoft** present

A CD-ROM designed and distributed to give you the opportunity to evaluate some of the most innovative Java development tools available.



MARCH 2002 EDITION



**sys-con**

**www.sys**



n media  
-con.com

SAVE 30% Off\*  
the annual  
newsstand rate

# JAVA DEVELOPER'S JOURNAL

\* Offer subject to change without notice

ANNUAL NEWSSTAND RATE
<del>\$71.88</del>
YOU PAY
\$49.99
YOU SAVE
30% Off the Newsstand Rate

## DON'T MISS AN ISSUE!

Receive 12 issues of *Java Developer's Journal* for only \$49.99! That's a savings of \$21.89 off the cover price. Sign up online at [www.sys-con.com](http://www.sys-con.com) or call 1-800-513-7111 and subscribe today!

Here's what you'll find in every issue of *JDJ*:

- Exclusive coverage
  - J2EE
  - J2SE
  - J2ME
- Exclusive feature articles
- Interviews with the hottest names in Java
- Latest Java product reviews
- Industry watch



# Using the JTable

Updating the  
database

WRITTEN BY BOB HENDRY

In Parts 1 and 2 of this article (*JDJ*, Vol. 6, issues 1 and 7) I discussed how to use a JTable with a table model and showed how much work is involved getting a JTable to work with data. This is quite a departure for veterans of other fourth-generation languages who may be used to developing in Visual Basic or PowerBuilder.

Both these languages have intelligent controls that keep track of the data as the user is manipulating it. These controls can then determine how to handle database changes such as inserts, updates, and deletes. Java doesn't have any built-in functionality. Remember the old Java adage: "To use it you must first build it."

Remember that the JTable or the table model is in no way connected to the database. Even when you're instantiating the JTable based on its table model, the model simply populates a collection (usually vectors), then passes them back to the JTable. Any necessary functionality must be programmed to use the JTable to perform actual database manipulation.

It would be nice if a JTable or its associated model had a method called Update(). Unfortunately, it doesn't, at least not yet. With a bit of work, by the end of this article you should be able to program such a method. Before I discuss the login needed for real database updates, I'll discuss the groundwork involved. Primarily, how can the JTable and table model be configured to detect user changes and how to add and delete rows. When these three tasks can be accomplished, only then can the database be updated.

I'll walk through the steps needed for database updates. Listing 1 provides the complete code, and Figure 1 displays the application (Listings 1-9 can be found on the *JDJ* Web site, [www.sys-con.com/java/sourcec.cfm](http://www.sys-con.com/java/sourcec.cfm).)

## Detecting User Changes

Where do we start? Before we can think about updating a database, the table model must first be aware of user changes. In case you haven't noticed, by default the JTable and associated table model don't apply any user-sup-

plied changes. For example, if a cell value is "Cheeseburger" and the user types "Hotdog" over it, the new value is displayed only when that current cell has the focus. As soon as the user tabs to another cell, the old value is restored. This is not a bug. Remember, the programmer is responsible for all behavior. The old value is restored because no code exists to say otherwise.

How can we get the new value to remain in the cell? By coding the setValueAt() method in the table model. This method from the TableModel class is automatically fired when the contents of the cell are modified and the focus is changed to another cell. This method tells us the row, column, and new value of the cell. Then code has to be written to update the data (in our case vectors) that make up the table model. In Listing 2, the vectors that make up the table model are updated with the new value for the cell.

## Adding Rows

Any good user interface has the functionality to add rows. Any GUI you write using a JTable should include it. But where can this functionality be added? Remember, the JTable is merely the view of the data. Most functionality, including the addition of rows, must occur in the table model. With that in mind, there should be a method called addRow() or insertRow(), for example, available for the table model, right? Guess again. Such methods must be programmed. If you think about it, the absence of such built-in Java methods makes sense. To understand why, you must first see what a "row" really is.

Because the table model contains the data for the JTable, it also controls and is aware of what a row looks like. The JTable is pretty oblivious to both these facts. A "row" in a table model can and will look very different from application to application. For example, Application 1 may have three columns with the data types string, integer, and boolean. Application 2 may have four columns with the data types string, string, float, and integer. The concept of a "row" really exists only for the beholder. As far as Java is concerned, a row is a vector of supporting classes. The data

types for these classes are as varied as the imagination of the programmer who created them. Because of this variation there's no built-in Java method to insert or add a row to a table model, because Java doesn't keep track of what a row looks like. This is the responsibility of the programmer.

In our example, a row in a table model is made up of a vector. Each element within the vector reflects the data type of the database column retrieved into it. To add or insert a row, the vector must first be queried about what data types it contains. These data types can then be built and added into another vector. This resulting vector can then be added to our table model, effectively adding a row. Listing 3 demonstrates how to add a row to the table model. For brevity, only vectors containing strings, integers, and booleans can be added.

### Deleting Rows

Fortunately, deleting rows in the table model is a bit less challenging. Deleting is fairly easy because it's irrelevant which data types the row consists of. The only real concern is to remove the row from the table model. However, the deleted row needs to be remembered in some way when we try to delete it from the database.

When the database is updated, SQL DELETE statements will have to be built. Even when the row no longer exists in the table model, the primary key for the deleted row must be remembered so the corresponding record in the database can also be deleted. This is accomplished by saving the primary key for the deleted row into a vector.

Later, when we build the SQL DELETE statement, we can ask the vector to tell us which row to delete. For simplicity, Listing 3 assumes the primary key is numeric and is the first column in the row. With a bit of ingenuity, this functionality can be expanded to include any number of columns with any data type. Listing 4 illustrates how to delete a row from a table model, but remember its primary key. This simple method deletes a row from within the vector and notifies the JTable to update the view – making the row removal visible to the user.

### Updating the Database

Now for the fun part. So far our table model can handle data modifications, new rows, and deleted rows. The next step is to apply the changes to the database. Since the table model and the database don't know about each other, it's up to the programmer to determine how the database will be affected. The concept of applying changes to the database is simple – create and execute a SQL statement. Depending on the status of the row in the table model, a SQL INSERT, DELETE, or UPDATE statement needs to be built. Sounds easy? Well, it is. The hard part is writing the code that will build the SQL. Once the code is built, database transactions are a snap. The next section will discuss and demonstrate the code needed to generate SQL statements.

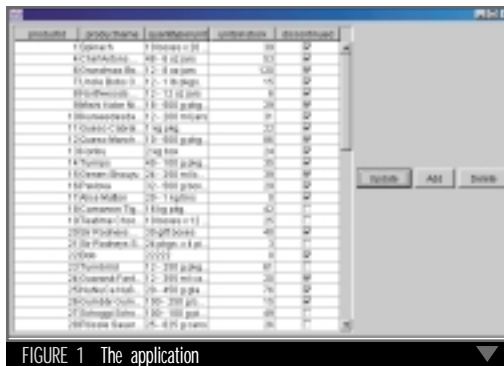


FIGURE 1 The application

### Groundwork

Before the code can be written to generate SQL, a few housekeeping chores are in order. Information such as the primary key, names of the columns in the database, and a user-entered value from the table model must be obtained. The primary key, as well as the database column names, can be retrieved through Java metadata methods. For simplicity, this table model will assume that the first column in each row is the primary key and the data type is numeric. Listing 5 keeps track of how many rows are in the table model, what the user-entered values are, and the value for each of the primary keys for each row.

At this point code needs to be written that will perform two loops. First, the code loops through each row of the table model, then through each element within the row. Remember that rows within the table model are really collections of other objects (e.g., strings and integers). Each element within the row must be

# SUBSCRIBE AND SAVE

## XML JOURNAL

Offer subject to change without notice

ANNUAL NEWSSTAND RATE	
<del>\$93.88</del>	
YOU PAY	
\$77.99	
YOU SAVE	
\$5.89	Off the Newsstand Rate

### DON'T MISS AN ISSUE!

Receive 12 issues of *XML-Journal* for only \$77.99! That's a savings of \$5.89 off the annual newsstand rate.

Sign up online at [www.sys-con.com](http://www.sys-con.com) or call 1 800 513-7111 and subscribe today!

### In February XML-J:

#### SOAP Messages with Attachments

How this emerging W3C note can be used with the Apache SOAP implementation

#### Applied Templates

Simplify XSLT code with applied templates

#### XML Development Environments

How to create a source environment for XML

#### Q&A: DIVE into XML

by Trace Galloway

#### I Am the Official Mascot of XSLT

A transforming robot looking for work

#### Intelligent XML Content Firewalls

Semantic-based filtering of digital content



SAVE 30% off the annual newsstand rate

BUSINESS & TECHNOLOGY  
**wireless**

Offer subject to change without notice

ANNUAL NEWSSTAND RATE
<del>\$71.88</del>
YOU PAY
<b>\$49.99</b>
YOU SAVE
<b>30%</b> Off the Newsstand Rate

**DON'T MISS AN ISSUE!**

Receive 12 issues of **Wireless Business & Technology** for only \$49.99! That's a savings of 30% off the cover price. Sign up online at [www.sys-con.com](http://www.sys-con.com) or call 1 800 513-7111 and subscribe today!

## Wireless Business & Technology:

**Going Wireless: An Introduction to Wireless Security**  
Kevin Wittmer provides an overview of security technologies available today for LAN and WAN wireless networks

**The World Phone: When Will It Truly Work?**  
A look at the primary competitors and what their strategies are vis-à-vis world phones

**Wireless Internet: The Next Generation**  
NTT SOFT is bringing their unified vision of wireless and Internet technology to the U.S.

**Boingo Jumps In:**  
This company has what it takes to kick-start the Wi-Fi service industry

**Product Technology:**  
Pocket PC Is Enterprise-Ready with a Built-in VPN Client



queried for its value, data type, and whether it contains the value for the primary key.

When testing each column for its data type, you can start saving the user-supplied values to be used in the SQL. Data type is very important because it changes the way the program keeps track of the user-entered values. For example, if the user changes a column of the data type STRING, the program must wrap single quotes around the value, otherwise the SQL statement will fail. Listing 6 queries the elements in each row for three different data types. For simplicity, this example doesn't test for all possible types; this modification is simple once you understand the basics.

### Performing the INSERT

Now that we have data describing each column – as well as the data itself

it, it's executed against the database. The record has now been updated.

### Performing the DELETE

Deletes work a little differently. Since the row no longer exists in the table model, there won't be any current row to loop through. When the row was deleted from the table model, the primary key of the row was saved in a vector. When deleting a row in a database, the column names and the data values are irrelevant. All that's needed is the primary key. Listing 9 obtains the key for deleted rows and builds a SQL DELETE statement. Also, for the sake of brevity, this code assumes that the record being deleted has no foreign key constraints. If it does, the DELETE statement would fail, of course.

### Final Considerations

Now that wasn't too bad, was it? The tricky part was creating the SQL

“ Any good user interface has the functionality to add rows. Any GUI you write using a JTable should include it

– code can be added to INSERT rows in the database, basically creating and executing a SQL INSERT statement. First, the vector that contains all the new row numbers is queried as to how many entries it contains. If it contains any entries, the row number in the vector is compared to the current row number in the table model. If the row matches, the SQL statement can be built. Listing 7 loops through all the columns and data values and builds the SQL INSERT statement. After the statement is built, it's executed against the database. If all goes well, the new row is now saved.

### Performing the UPDATE

If the current row has not been inserted, a SQL UPDATE statement is built. Listing 8 obtains the name of the database column as well as its value. After the UPDATE statement is creat-

statements. As a disclaimer I'd like to point out that the code in this example has been simplified in order to demonstrate the basics of how to use the JTable and associated table model to save changes to the database. For example, this article doesn't take into consideration SQL errors that can occur when violating a database constraint. Also, row management would be better served by using hashtables to hold the status of each row as well as the primary key value. In any event, the code is functional and generic enough for use in most projects. Feel free to use and improve it as you see fit. ☛

### AUTHOR BIO

Bob Hendry is a Java instructor at the Illinois Institute of Technology. He is the author of Java as a First Language.

[bobh@envisionsoft.com](mailto:bobh@envisionsoft.com)

EnginData Presents

# 2002 Developer Market Survey Reports



engindata.com

Our comprehensive reports offer insight and strategy to guide your most critical business decisions in today's fastest growing technologies...

- ✓ Establish your product and marketing strategy
- ✓ Understand your customer's needs
- ✓ Evaluate Technology & Trends

**engindata** ✓  
RESEARCH

Premiering...*this winter*

subscribe **Now!**

**FORFAST**  
DELIVERY

Go  
Online  
and  
Subscribe  
Today!

The World's leading  
Independent WebLogic  
Developer Resource

FOR WLS DEVELOPERS BY WLS DEVELOPERS  
**WebLogic**  
DEVELOPER'S JOURNAL  
An introduction to...

Helping  
you enable  
inter-company  
collaboration  
on a global scale

- Product Reviews
- Case Studies
- Tips, Tricks and more!

**SPECIAL**  
INTRODUCTORY OFFER  
**SAVE \$31\***  
HURRY, DON'T DELAY! OFFER EXPIRES DECEMBER 31, 2001

**WebLogicDevelopersJournal.com**

SYS-CON Media, the world's leading publisher of *i*-technology magazines for developers, software architects, and e-commerce professionals, brings you the most comprehensive coverage of WebLogic.

\*Only \$149 for 1 year (12 issues) regular price \$180.

**SYS-CON**  
MEDIA

# IBM AND MICROSOFT SLUG IT OUT

## WEBSPHERE VS .NET

**IBM RESPONDS TO LATEST MICROSOFT JABS AT WEBSPHERE®4.0. MICROSOFT "MISSING THE POINT," SAYS IBM PROGRAM DIRECTOR.**

**EXCLUSIVE** to *Java Developer's Journal Industry Newsletter*, here's the latest fiery rebuttal in the red-hot IBM vs Microsoft debate that's been raging on the Internet over the superiority of their respective platforms for creating Web services. Stefan Van Overtveldt, program director, WebSphere Technical Marketing, IBM, holds that Microsoft's original white paper belittling WebSphere 4.0 was fatally flawed from the start due to its premise, which in his words, "is missing the point." Before Van Overtveldt's complete response, *JDJ-IN* reviews highlights of the verbal battle's progress to date:

Microsoft launched the initial salvo in the Internet war, with a white paper that compared creation of Web services (using the PetStore.com scenario) using Visual Studio.NET versus IBM WebSphere v4.0. To support their claim that .NET has a significant advantage over WebSphere, Microsoft hired an independent consulting firm to develop a Web service using both its own platform and IBM's. According to Microsoft, the results of this benchmarking exercise proved .NET the winner in developing Web services.

IBM responded with its WebSphere competitive review (see [www3.ibm.com/software/info1/websphere/news/ibmnews/compreview4.jsp?S\\_TACT=101CMW13&S\\_CMP=campaign](http://www3.ibm.com/software/info1/websphere/news/ibmnews/compreview4.jsp?S_TACT=101CMW13&S_CMP=campaign)), calling Microsoft's white paper "misleading," and firmly stating that "there is no doubt that WebSphere is the superior platform for developing Web services." IBM pointed out that Microsoft's study used the IBM Web services Toolkit, when the Web service should have more appropriately been built using the new WebSphere Studio Application Developer tool. IBM also said Microsoft overstated by nearly six hours the amount of time needed to create the service with IBM WebSphere. Furthermore, IBM said that .NET needed 106 lines of handcrafted code compared to 1 line in WebSphere Studio, and that total cost for constructing the Web service was lower with IBM.

Microsoft returned the volley with a "Response to IBM" (at [www.gotdotnet.com/team/compare/ibmrespond.aspx](http://www.gotdotnet.com/team/compare/ibmrespond.aspx)). Microsoft's reply minced no words: "IBM is attempting to mislead customers," it states, in a point-by-point comparison of IBM's claims and Microsoft's positions on issues including cost of deploying the service, the number of lines of code needed to build and consume the Web service, the requirement of BizTalk server, the standards adhered to, and .NET's ability to work in mixed environments. Microsoft did concede that IBM's new WebSphere Studio Application Developer tool improves IBM's support for building Web services, and Microsoft as a result issued a completely updated white paper comparing its product with that version (see [www.gotdotnet.com/team/compare/webservicecompare.aspx](http://www.gotdotnet.com/team/compare/webservicecompare.aspx)). Microsoft has more background on its claims for the superiority of Visual .NET at <http://msdn.microsoft.com/net/compare/default.asp>.

**"THERE'S ONLY ONE COMPANY RIGHT NOW THAT OFFERS A COMPLETE WEB SERVICES INFRASTRUCTURE, AND THAT'S IBM"**

—STEFAN VAN OVERTVELDT, IBM

**FOR IBM'S COUNTERREPLY, AN INTERVIEW WITH STEFAN VAN OVERTVELDT, PROGRAM DIRECTOR, WEBSPHERE TECHNICAL MARKETING, IBM**

**Stefan Van Overtveldt:** The premise of starting off with saying, "Let me show you how much more productive our development tools are by re-creating the PetStore.com application in C# and with Visual Studio .NET," is beside the point. PetStore.com is an application that's been written to allow J2EE application vendors to test if all of the J2EE APIs are actually present and functioning well in their J2EE application server. That's the only objective that PetStore.com ever had. This being said, because it does go out and test all of those different APIs, this is an application that is not well written at all. It's not well written for performance. It's not well written for security. It is just a test. Taking this application and pretending that it's a real live application, and that the customers can draw conclusions with regards to productivity, performance, etc., is missing the point about what this application was intended to be.

When you look at trying to enable the application as a Web service, not just from a development perspective but also from a deployment perspective (because, by the way, customers do want to deploy these Web services), we are still confident that we have tremendous productivity gains over any other application developer in the industry. If you look at what Web services are really all about, they're not just about just taking an application and making it available through an XML SOAP interface or to a WSDL wrapper, etc. You have to look into what is this going to do towards your entire infrastructure, what's the impact of opening up an application to the outside world, which is basically what you're doing. The impact is that you need to put in stronger security mechanisms. You need to be sure you can handle the workload. You need to be sure that you can quickly leverage these Web services as part of existing applications or expose existing applications as Web services. It's a much bigger picture than just taking an application and publishing it out as Web services.

You can debate different ways of doing this, but there's only one company right now that offers a complete Web services infrastructure, and that's IBM. We have Web services supported in our application servers. We have it in our development tools. We have a Web services infrastructure with regards to private UDDI gateways, for example, or private UDDI registries and UDDI gateway functionality. We have ways of managing Web services in a secure environment with a product like Tivoli Manager for Web services. We even have a number of technologies on the table that customers can use to take existing applications, not just stuff that was newly developed in C# or J2EE, but existing applications, applications that they've been using for years. Things like their SAP ERP systems, database applications, transactional applications running on IBM mainframe platforms, etc., and make those available very rapidly as Web services, again in the same managed and controlled environment. That is something no other company can offer.

**JDJ-IN:** The use of the PetStore setup did seem problematic.

**SVO:** We can argue about which company or which tool is more productive in this scenario for years to come. Truth is, it's beside the point. It's not a representative application. It does not make a lot of sense to go out and re-create an application in another language and see if it performs better. If I were to rewrite this application, and believe me I'm not a good programmer, it would probably perform better.

**JDJ-IN:** One issue aside from performance that Microsoft was slamming IBM on was cost. IBM's first response to Microsoft said a 4-server deployment using Microsoft .NET would cost about \$399,996. Microsoft's response to that was IBM "is just wrong. IBM is attempting to mislead customers." Microsoft

# **sys-con media**

[www.sys-con.com](http://www.sys-con.com)

went on to say that WebSphere 4.0 costs over 15 times more than Microsoft .NET for this typical clustered scenario.

**SVO:** Well, if nobody is accessing the cluster, they're probably right. The WebSphere licensing is based on buying an application server, and we have a per-processor licensing scheme, and you can put those applications, it's \$12,000 list price per processor, in a cluster. So, if you have a 4-node cluster, for example, that would cost you again at list price, \$48,000. Throw in 10 developer seats around and you're looking at a total solution of around \$80,000. But that is it. Whether that solution is serving one concurrent user or it's serving 100,000 concurrent users, the price remains the same. We don't have the notion of access licensing. If you count how the developer license works, if you count on actually putting those applications in production, not just showing a demo of it, you have hundreds, thousands of other applications out on the Internet accessing that Web service, then the Microsoft solution is a lot more expensive. It's just because it's a completely different way of counting licenses.

**JDJ-IN:** That's what is accounting for the cost differential: how they're approaching their cost basis?

**SVO:** This is the same thing that goes back to Pet Store. Pet Store is a lab type of application, something you would never put in production. And that has a number of consequences. If you set up this type of environment, in a lab environment, to calculate what it costs you, maybe Microsoft has a point. Maybe they don't. But if you put it into a production environment where you have to start accounting for a number of concurrent accesses to these servers, we are very sure that our solution is much more cost-effective.

**JDJ-IN:** The last point Microsoft emphasized in their most recent response had been regarding code. Microsoft said that to create a PetStore Web service, IBM WebSphere Studio Application Developer required 82 handcrafted lines of code and Visual Studio .NET required 48, about half; and to create simple client to consume Pet Store Web service, IBM required 49 lines of handcrafted code; Visual Studio .NET, 24. Now is that still a function of Pet Store's being atypical of a real-life application code-writing situation?

**SVO:** Again, I did not personally run those tests. The one point I can make is that most of the ability we have in Web Studio Application Developer is to take an existing J2EE application and render it as Web services fully automatic. Now is there some hand coding involved and is it 48/24 lines of code, double in our scenario? I don't know. What I do see is that taking this application and making it available as Web services is, again, only part of what you need to do. Because you need to link this application to a security environment, which Pet

Store does not do, by the way. You need to link this application to a management environment, which Pet Store does not do. With WebSphere Studio Application Developer we can do all of those things pretty much in an automated fashion. You may need to write some line of code here and there, but that is a much more realistic scenario to look at. I am absolutely convinced that if you look at what are the real overall requirements that you actually need to put a Web service in production our development tools are much more effective and much more productive than any other tool out there.

The larger question is, which platform are customers choosing to deploy Web services? Giga Information Group just issued a report saying that J2EE platforms are the big winners overall among early adopters of Web services technology and the most important to Web services strategy. When it comes to companies that are actually evaluating and deploying this, Giga says IBM WebSphere is the clear favorite over Microsoft. In the end, it is the market that is deciding, and according to Giga, customers favor WebSphere over .NET.

## **JDJ-IN Exclusive: Microsoft Replies to IBM**

### **MICROSOFT CALLS FOR AN "INDEPENDENT SHOOTOUT" TO RESOLVE .NET VS WEBSHERE PERFORMANCE COMPARISONS**

*By Gregory Leake  
Group Product Manager, Microsoft Corporation*

**I AM WRITING** about your article titled "IBM Responds to Latest Microsoft Jabs at WebSphere 4.0." In this article, you interview IBM's Stefan Van Overtveldt, program director, WebSphere Technical Marketing.

In the interest of facts and truth, I would like to correct Mr. Van Overtveldt on a couple of points, and present a Microsoft response to his interview and the article.

First, I want to point out that the .NET Pet Shop comparison to the Java Pet Store is separate from the Web services comparison between .NET and IBM WebSphere 4.0. Both comparisons can be found at <http://www.gotdotnet.com/team/compare>. The article and the interview discuss both comparisons as if they are the same.

So let's separate them and look at the bulk of what Mr. Van Overtveldt says about each.

1. With regard to the .NET Pet Shop comparison, he says that:

"PetStore.com is an application that has been written to allow J2EE application vendors to test if all of the J2EE APIs

**RECEIVE \$150**  
DISCOUNT OFF FULL CONFERENCE  
WEB SERVICES EDGE REGISTRATION

web services **EDGE**  
conference & expo

2002 WORLD TOUR

**Learn How to Develop  
SOAP Web Services NOW!**  
at a One-Day Tutorial... Coming to a City Near You!



are actually present and functioning well in their J2EE application server. That's the only objective that PetStore.com ever had.... this is an application that is not well written at all. It's not well written for performance. It's not well written for security.... Taking this application and pretending that it's a real live application, and that the customers can draw conclusions with regards to productivity, performance, etc., is missing the point about what this application was intended to be."

This is wrong. The Java Pet Store (Petstore.com) is held up by Sun Microsystems as a primary blueprint application for J2EE that illustrates "best practice architecture" and "best coding practices...for enterprise applications." It has an entire Sun blueprint Web site dedi-

various vendors involved. Right now, the .NET Pet Shop remains uncontested as far as published versions of the Pet Store go, with 1/4 the amount of code required to build vs the Java version, and offering 28 times better performance and over 8 times better scalability. We are very happy to meet IBM, Oracle, or other J2EE vendors in an independent shootout for verification of the performance results.

2. With regard to our .NET vs IBM Web services comparison, the bulk of the interview seems to focus on the license cost comparison. Here again, Mr. Van Overtveldt is misleading customers, since he claims that our cost comparison does not take into account client access fees associated with the deployment. I would like to point out that we fully take into account client access licenses in the cost

## “IBM SHOULD CORRECT THEIR PUBLIC DOCUMENT BECAUSE IT IS WRONG”

—GREG LEAKE, MICROSOFT

cated to it as a sample application for customers to follow. Sun has even published a Sun Blueprint-series book all about the Pet Store as a design pattern for enterprise applications, and has been telling enterprise developers to follow it for building scalable, reliable applications since JavaOne in May 2001. Furthermore, IBM appeared on stage with Sun and other J2EE vendors to endorse the Pet Store application at JavaOne in May 2001, and even demonstrated it running in WebSphere 4.0 in front of thousands of developers as a best practice enterprise application. And furthermore, they ship it in the IBM WebSphere 4.0 product as a sample application for developers to follow. So it is very disingenuous for IBM to come out now and say it is a bad application and should not be used to compare with .NET. They are all of a sudden singing a very different tune based on the release of the .NET Pet Shop.

Sun has also now come out and said that the design pattern is valid, but it is not designed to be high performance. In response, we find it a highly questionable customer practice to publish a "best practice enterprise design pattern" and not ensure it will result in high performance applications. In short, if this is the case, shame on Sun for publishing it and promoting it as an enterprise design pattern to begin with. They should either rewrite it, pull it from the Web and their blueprint series published books, or stand behind it.

As for the benchmark of the Pet Store application, I would like to point out that the benchmark was originally conducted by Oracle as part of their Java Performance Challenge. So in this way a major J2EE vendor invited the comparison. In the end, I think the comparison is quite valid, and the competition is healthy for customers and the

comparison, and that he is simply misinformed on this topic. Mr. Van Overtveldt claims a 4-server WebSphere deployment would cost a fixed amount no matter how many clients access the WebSphere server. He states that the MS cost, however, may be lower out of the gate, but would be higher and grow depending on the number of clients accessing the site. This, he claims, is because the client access licenses required for Windows 2000 Server drive up the cost for .NET as more users connect.

This is wrong. In fact, we have included a Windows 2000 Server Internet Connector License in our cost calculation, which includes \*unlimited\* authenticated client access to the .NET Web Service in question. So we fully stand behind our cost comparison, no matter how many users are connected and using the Web service. For a 4-server deployment with 8 CPUs per server, IBM WebSphere 4.0 would cost \$12,000 per CPU or a total of \$384,000. Microsoft .NET would cost \$3,999 (W2K Advanced Server) + \$1,999 (Internet Connector License) = \$5,998 per server for a total of \$23,992, no matter how many clients use the W2K site. Mr. Van Overtveldt should make a point of understanding the MS licensing cost before misrepresenting it, and IBM should correct their public document because it is wrong. IBM WebSphere 4.0 costs 15 times more than Microsoft .NET in the example analyzed, which is very typical of a real customer deployment configuration in the enterprise. ☛

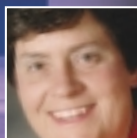
To add your comments to the discussion, go to [www.sys-con.com/java/](http://www.sys-con.com/java/).

WebSphere® is a registered trademark of IBM Corp. and is used by SYS-CON Media under license.

# Jump-start your Web Services knowledge Get ready for Web Services Edge East and West!

AIMED AT THE JAVA DEVELOPER COMMUNITY AND DESIGNED TO EQUIP ATTENDEES WITH ALL THE TOOLS AND INFORMATION TO BEGIN IMMEDIATELY CREATING, DEPLOYING, AND USING WEB SERVICES.

EXPERT PRACTITIONERS TAKING AN APPLIED APPROACH WILL PRESENT TOPICS INCLUDING BASE TECHNOLOGIES SUCH AS SOAP, WSDL, UDDI, AND XML, AND MORE ADVANCED ISSUES SUCH AS SECURITY, EXPOSING LEGACY SYSTEMS, AND REMOTE REFERENCES.



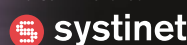
### PRESENTERS...

**Anne Thomas Manes, Systinet CTO**, is a widely recognized industry expert who has published extensively on Web Services and service-based computing. She is a participant on standards development efforts at JCP, W3C, and UDDI, and was recently listed among the Power 100 IT Leaders by Enterprise Systems, which praised her "uncanny ability to apply technology to create new solutions."



**Zdenek Svoboda is a Lead Architect** for Systinet's WASP Web Services platform and has worked for various companies designing and developing Java and XML-based products.

EXCLUSIVELY SPONSORED BY



**BOSTON, MA** (Boston Marriott Newton) ..... **JANUARY 29**  
**WASHINGTON, DC** (Tysons Corner Marriott) ..... **FEBRUARY 26**  
**NEW YORK, NY** (Doubletree Guest Suites) ..... **MARCH 19**  
**SAN FRANCISCO, CA** (Marriott San Francisco) ..... **APRIL 22**

REGISTER WITH A COLLEAGUE AND SAVE 15% OFF THE \$495 REGISTRATION FEE.

Register at [www.sys-con.com](http://www.sys-con.com) or Call 201 802-3069



**sys-con**

[www.sys](http://www.sys)

**media**

-con.com

# 'How's the Boss?'

What's *your* supervisor's management style?

WRITTEN BY  
BILL BALOGLU &  
BILLY PALMIERI



Whether you're working for a small startup or a huge corporation, the quality of your daily life can be ruled (or ruined) by the style and personality of the person who signs your time card.

We spoke with several professionals who've had a wide variety of experiences with different types of managers. Here, categorized for your reference and convenience, are profiles of some of the most common types of managers and some advice on how to deal with them.

## Manager Profiles

### The Blame-Thrower

Described by one engineer as "the worst manager I've ever worked with," this manager lives in fear that a team member will claim or steal credit for every task the group accomplishes. He or she will always try to blame somebody on the team for any mishap that occurs.

The common motivator that drives these managers is fear – fear of higher management and fear of their own staffs. The worst thing about this type of manager is that they never accept the basic responsibility of a leader.

- **How to deal with a Blame-Thrower:** It's not easy! Think of this person as someone who was put on this Earth to test your professionalism – and your patience. Try your best to establish a good rapport with this manager (and others in the group). Take the high road in all your dealings with this person and don't badmouth him or her to other team members (it can come back to bite you). Good luck!

### The Micro-Manager

This type of manager is familiar to people in all industries: the boss who at some level is convinced that the staff is basically incompetent and therefore

**W**e've all had them. Some of us have been them. But if there's anything that seasoned engineers have in common, it's lots of experience working with different kinds of managers.

requires constant double-checking and supervision.

The discouraging message that the Micro-Manager conveys is a lack of respect for the basic talent and ability of those under his or her supervision.

- **How to deal with a Micro-Manager:** (See above.) Don't take the bait and take micromanaging personally; chances are this manager treats everyone this way. When frustrated, do your best to remind yourself it's only a job.

### The Do-It-All Manager

On the opposite end of the spectrum from the Micro-Manager is the manager who may have been promoted to a position of authority because he or she was so strong as an individual contributor.

The problem with the Do-It-All Manager is an inability to delegate authority, which also implies a lack of trust and respect for his or her staff. This manager is typically overworked, believing that no one else can do the work as well, while the team members are underused and become bored by the basic, unchallenging tasks on their plate.

- **How to deal with a Do-It-All Manager:** Prove your professionalism and your abilities on a regular basis. And politely offer to take on more challenging and additional work.

### The Mentor

This type of manager is secure enough in his or her own abilities to take on the responsibility of training and supporting the team.

A truly secure Mentor will share past mistakes as well as successes with team members. Others will share only their successes.

- **How to deal with a Mentor:** Thank your lucky stars. Be grateful to be

working for someone who cares about your professional development.

### The 10-Steps-to-Success Motivator

This type of manager is eager to motivate the team with words of wisdom that often sound like they were lifted from a book, seminar, or series of motivational tapes.

What this Motivator doesn't realize is that advice earned from experience has more meaning for the staff than canned advice.

- **How to deal with a 10-Steps-to-Success Motivator:** Give some of the techniques a shot. Even if they seem canned, some of them might be helpful.

### The Leader

This manager is respected by the team because he or she has a thorough understanding of the technical aspects of the project. The Leader challenges the staff with aggressive goals, clear deadlines, and a fundamental belief in the team's abilities.

The Leader has enough confidence in his or her own abilities to give company-wide credit and visibility to team members for their individual accomplishments.

- **How to deal with a Leader:** If you're lucky enough to work for a true Leader, make it known how much you appreciate his or her management style. Those who give credit where it's due also appreciate it themselves.

### Who Did We Miss?

Have we left anyone out? Have you developed any successful strategies for working with these types of managers? If so, let us know. ☛

jdcolumn@objectfocus.com

## AUTHOR BIOS

Bill Baloglu is a principal at ObjectFocus ([www.ObjectFocus.com](http://www.ObjectFocus.com)), a Java staffing firm in Silicon Valley. Previously he was a software engineer for 16 years. Bill has extensive OO experience and has held software development and senior technical management positions at several Silicon Valley firms.

Billy Palmieri is a seasoned staffing industry executive and a principal at ObjectFocus. His prior position was at Renaissance Worldwide, a multimillion-dollar global IT consulting firm, where he held several senior management positions in the firm's Silicon Valley operations.

# sys-con media

[www.sys-con.com](http://www.sys-con.com)

## Next Month

### EXTENDING THE J2SE 1.4 LOGGING PACKAGE

Monitoring made easy  
by Jim Mangione

### JINI SURROGATE AS A PLATFORM FOR J2ME GAMES

Surrogate architecture incorporates smaller devices  
by William Swaney

### MAC OS X & JAVA

A perfect marriage  
by Hitesh Seth

### BOOK REVIEW:

Java Internationalization  
by David Czarnecki and Andy Deitsch  
reviewed by Ajit Sagar

### JAVA AND WIRELESS

Building end-to-end Palm applications using Java  
by JP Morgenthal

### INTEGRATING J2ME, GPS, AND THE WIRELESS WEB

by Shane Isbell

**JAVA** DEVELOPERS  
JOURNAL

**Special online offers** SAVE UP TO \$100  
ON MULTIPLE SUBSCRIPTIONS

Pick **4** or **5** and Subscribe  
for one **special low price**



**SYS-CON**  
MEDIA

[WWW.SYS-CON.COM/2001/SUBOFFER.CFM](http://WWW.SYS-CON.COM/2001/SUBOFFER.CFM)

Offer subject to change without notice

# Brushes with Greatness



WRITTEN BY  
BLAIR WYMAN

**F**or those of you who have been following “Cubist Threads” from its inception, you know – both of you – that several of my little musings have centered around brushes I’ve had with “greatness.” Presuming I understand my own thought processes well enough to comment, I’d have to guess that recounting these events has been little more than a failed attempt to mask my own mediocrity (or, perhaps, to emerge from it).

So far, my recollections have all come from my years before IBM. Oh, it’s not that I haven’t encountered greatness here – quite the contrary – it’s just that you probably wouldn’t recognize most of the names. For instance, who’s ever heard of Benoit Mandelbrot?

Okay, savvy computer folks, that was a rhetorical question. Unless you’ve been living under a heat sink for the past 15 years, you’ve undoubtedly heard of the so-called “Mandelbrot Set,” named after this august IBMer. Creative renderings of the “edge” of that eminent region of the complex plane might well adorn your screen or wall or necktie.

The Mandelbrot Set holds a special place in my personal computing history: I probably owe my early career to its simplicity, beauty, and allure. Long after my introduction to his work, I actually met Benoit Mandelbrot at a conference in 1989; hence, this month’s “brush with greatness.”

I dabbled briefly with computers in high school, but the ugliness of the whole “batch” programming process – punch your card deck, submit it to the operator, pick up your results, and try to think up new epithets when you realize you’ve wasted two hours – left me cold. I’ve always been a sort of “instant gratification” person at heart, so mid-’70s batch technology and I were fundamentally incompatible.

Then, after a time, computing and I met again. While I had changed quite a bit during the intervening years, computing had changed even more dramatically. The year was 1985, the personal computer was a reality, and I bought my first – a PC-XT, with its incredible 10 megabytes of hard disk space – to help me pursue my degree.

Trying to use the version of BASIC that came with my XT was almost my undoing. I remember studying the BASIC manual, thinking I’d never make any progress. “Hello, world” was about as far as I was getting, and it was frustrating. Then I discovered an amazing, affordable piece of software that was a true work of art, Turbo Pascal.

TP version 3 included a complete editor, compiler, and debugger, cost about \$35, and required only 37K of runtime RAM. Further, it could run circles around anything else I could afford (which wasn’t much). Basically, I got “hooked” into programming by Turbo Pascal, and have never been the same.

What does this have to do with the Mandelbrot Set? My introduction to this mathematical marvel came in the form of a fine-print program listing among the pages of *Turbo Technix*, a short-lived Turbo-centric publication from Borland. Included with an article about the Mandelbrot Set was the complete source code for a simple renderer.

I pored over the pages, typing carefully, and finally got a clean compile. I didn’t know it at the time but my life had just changed forever.

Every night before going to bed I’d pick out a new region of the set to render. The PC would labor in silence all night long, and when I got up I would be greeted by yet another adventure into some corner of infinity. Of course, after spending a few minutes enjoying the monochrome green-on-black rendering, I’d pick a new region and kick off the process again.

For those of you who haven’t dipped your toes into the infinite depths of the Mandelbrot Set – or some other fractal bounding main – a part of me envies you. I’ve spent many happy hours exploring the convoluted “reality” of the Mandelbrot Set, finding beauty in measures that only nature can provide. However, while shadows of its shape are realized with gates and CPUs and pixels, its sheer existence is independent of any such realization. The Mandelbrot Set is a fact of nature, and a beautiful mystery as well.

If you’ve never seen the Mandelbrot Set, or maybe just not lately, take a few minutes and go exploring. Point your search engine to the name “Mandelbrot” and take a peek at the folded edge of infinity. ☘

[blair@blairwyman.com](mailto:blair@blairwyman.com)

## AUTHOR BIO

Blair Wyman is a software engineer working for IBM in Rochester, Minnesota, home of the IBM iSeries.



**webgain**  
[www.webgain.com](http://www.webgain.com)

# sitraka

[www.sitraka.com](http://www.sitraka.com)