

# NEW SECURITY FEATURES IN JDK 1.2

U.S. \$4.99 (Canada \$6.99)

# JAVA™ DEVELOPER'S JOURNAL

JavaDevelopersJournal.com

Volume: 3 Issue: 5

## Easy Does It

by Rhett Guthrie pg.5

## Tips & Techniques Static Initializers & Uninitializers

by Brian Maso pg. 66

## The Grind

### Java Skeptics Run Amuck

by Java George pg. 82

## Under the Sun

### The Past, Present & Future of JFC

pg. 40

## CORBA Corner

### Developing Distributed Applications

by Qusay Mahmoud pg.52

## Product Reviews

### TopLink

by Ed Zebrowski pg.46

...

### Instant Basic for Java

by Ed Zebrowski pg.51

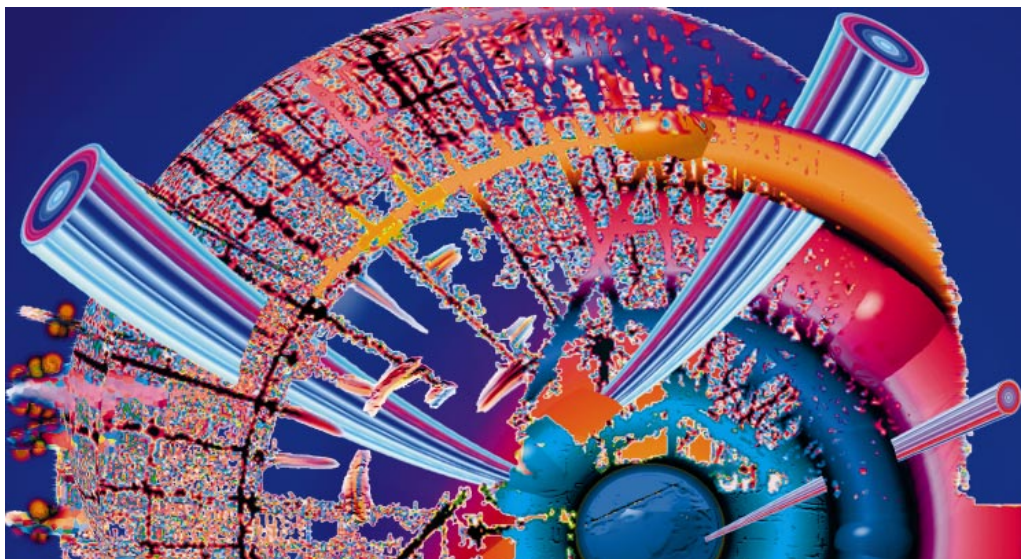
...

### JViews

by Ed Zebrowski pg.70

## Java News

pg.78



## JDJ Feature: A Generalized Enumeration Mechanism

*Developing complex iterators*

Myung Ho Kim

8

## Dealing with Network Timeouts in Java

*Making the case for setting socket options for timeouts*

David Reilly

64

## JDJ Security Feature: Protection Domains

*Building a comprehensive security package into the JDK*

Jahan Moreh

16

## JDJ Feature: Enterprise Strategy with Java

*Increased leverage along with easier building and debugging*

Graham Harrison

34

## The Cosmic Cup: Focus on the Java Platform

*Roles played by APIs that make up the Java Platform*

Ajit Sagar

58

## Legal Java: Java & the Law

*A legal discussion of Java development and patent laws*

Michael Zarrabian

72

## Reflection & Introspection: Object Exposed

*Developing useful utilities for Java development*

Ajit Sagar &  
Israel Hilerio

24

# Full Page Ad

# Full Page Ad

# Full Page Ad

## EDITORIAL ADVISORY BOARD

Ted Coombs, Bill Dunlap, Allan Hess,  
Arthur van Hoff, Brian Maso, Miko Matsumura,  
Kim Polese, Richard Soley, David Spenhoff

*Art Director:* Jim Morgan  
*Executive Editor:* Scott Davison  
*Managing Editor:* Anita Hartzfield  
*Associate Editor:* Christy Wrightington  
*Editorial Assistant:* Carolyn Emmett  
*Technical Editor:* Bahadır Karuv  
*Visual J++ Editor:* Ed Zebrowski  
*Visual Café Pro Editor:* Alan Williamson  
*Product Review Editor:* Jim Mathis  
*Games & Graphics Editor:* Eric Ries  
*Tips & Techniques Editor:* Brian Maso  
*Java Security Editor:* Jay Heiser

## WRITERS IN THIS ISSUE

Rhett Guthrie, Graham Harrison, Israel Hilerio,  
Myung Ho Kim, Qusay Mahmoud, Brian Maso,  
Jahan Moreh, David Reilly, Ajit Sagar,  
Michael Zarrabian, Ed Zebrowski

## SUBSCRIPTIONS

For subscriptions and requests for bulk orders,  
please send your letters to Subscription Department

**Subscription Hotline: 800 513-7111**

Cover Price: \$4.99/issue.

Domestic: \$49/yr. (12 issues) Canada/Mexico: \$69/yr.

Overseas: Basic subscription price plus air-mail postage  
(U.S. Banks or Money Orders). Back Issues: \$12 each

*Publisher, President and CEO:* Fuat A. Kircaali  
*Vice President, Production:* Jim Morgan  
*Vice President, Marketing:* Carmen Gonzalez  
*Advertising Manager:* Claudia Jung  
*Advertising Sales:* Paula Horowitz  
*Advertising Assistant:* Erin O'Gorman  
Jaelyn Redmond  
*Accounting:* Ignacio Arellano  
*Senior Designer:* Robin Groves  
*Designer:* Alex Botero  
*Webmaster:* Robert Diamond  
*Senior Web Designer:* Corey Low  
*Customer Service:* Rae Miranda  
Sian O'Gorman

## EDITORIAL OFFICES

SYS-CON Publications, Inc.  
39 E. Central Ave., Pearl River, NY 10965  
Telephone: 914 735-1900 Fax: 914 735-3922  
Subscribe@SYS-CON.com

JAVA DEVELOPER'S JOURNAL (ISSN#1087-6944) is  
published monthly (12 times a year) for \$49.00 by SYS-CON  
Publications, Inc., 39 E. Central Ave., Pearl River, NY 10965-2306.  
Application to mail at Periodicals Postage rates is pending at  
Pearl River, NY 10965 and additional mailing offices.

POSTMASTER: Send address changes to:

JAVA DEVELOPER'S JOURNAL, SYS-CON Publications, Inc.,  
39 E. Central Ave., Pearl River, NY 10965-2306.

## © COPYRIGHT

Copyright © 1997 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 coordinator.  
SYS-CON Publications, Inc. reserves the right to revise, republish and authorize its  
readers to use the articles submitted for publication.

ISSN # 1087-6944

DISTRIBUTED in USA by

## International Periodical Distributors

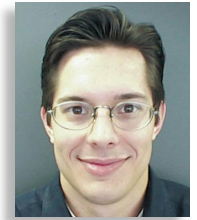
674 Via De La Valle, Suite 204, Solana Beach, CA 92075 619 481-5928

## BPA Membership Applied For August, 1996

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.



Rhett Guthrie



# Easy Does It

Visual Basic is arguably the most successful programming language in the history of programming languages. The number of VB components and applications out there is staggering, and the number of VB programmers is even more so. However, there is a not so silent contender for the World's Most Popular Language. It's OO, multithreaded and Internet-ready. It's an expressive and flexible language capable of industrial-strength server-side computing and, for the C++ crowd, here's the real rub: it's idiot-proof. It's Java. Java not only promises enterprise solution-capable software, it promises to do so with VB-style ease of use and with an unrivaled adoption rate. Therein lies the central issue: The combination of ease of use, power and popularity makes Java an important language for the software engineering community.

Consider distributed computing. There has been an evolution of technologies including socket programming, RPC and distributed objects. For a while, however, we had distributed objects with a little baggage – you had to muck with your application logic and you had to generate stub and skeleton code. This model is rapidly being eclipsed in favor of easier and more seamless programming models. Clients of remote objects often need to know that the server object is remote to be prepared to deal with the inevitable “gotchas” of network computing. However, there is no justification for coupling the server object to the distribution layer – it need not know how it's being accessed. Therefore, some distributed computing platforms are removing this burden from the programmer and allowing any class to be remote-enabled without modification. In fact, even the tedious and error-prone proxy class generation step now happens on demand at runtime. It no longer requires intervention from the programmer. What does this mean? Distributed computing in Java is approaching the theoretical limits of ease of use!

The beauty of this is that it's just the beginning. Distributed computing is just one tool in a software engineer's bag of tricks. The entire gamut of software engineering is subject to this level of ease of use. Java's thread model has almost made platform-independent, multi-threaded application development a non-issue. Garbage collection almost makes memory management a non-issue. Dynamic class loading greatly facilitates mobile agent platforms and applications. JavaBeans™ is quickly making GUI development a matter of connecting the dots. EJB promises to work similar magic for server-side transactional programming and persistence integration. JECF is on the way to solving the problem of developing electronic commerce software. The list goes on and on. Many of yesterday's programming nightmares are evaporating before our very eyes. As layer upon layer is

added, we'll find more software development issues being taken care of automatically. Expect it and demand it.

Microsoft Windows NT 5.0 is supposedly comprised of no less than 25 million lines of code. That is significant by any standard. Now, imagine if the authors had to write this opus, not in C and C++, but in binary. Imagine the complexity of such a task. The number and quality of minds needed would be tremendous. You could make a strong case that human sociological development has not yet advanced to the stage that such a group project is possible. It would seem that an accomplishment like NT 5.0 is simply not feasible (perhaps not even possible) without the higher levels of abstraction provided by C, C++, COM and the rest. My own experience in building distributed computing technology provides at least a modicum of evidence that Java, with its flexibility, ease of use and power, has the potential to go even further. Java makes it possible to achieve instant distributed computing, and there is every indication that there is much more to come.

Let's step back and consider the big picture. Why do we even care that software is easy to build? Does it matter that Java technologies are easy to use? Definitely, because by making it easier for us to build software, we are improving our ability to solve problems. The human race advances by the number of operations it can perform without thinking about them. When Alfred North Whitehead said this, he probably wasn't thinking about software engineering abstractions, but his words couldn't be more applicable. Let's face it, software runs the world. Many improvements in the nature of medicine, government, quality of life, science and economy can be directly linked to improvements in software. By making it easier to build software, we're making it easier to advance as a people.

The last thing the Java community needs is more hype. It is certainly not my point to compound the hype problem. My contention is that Java and the emerging frameworks do, or at least can, facilitate software engineering better than the commercially viable alternatives on the market. I offer the ease and growing adoption of distributed computing in Java as a success story. Java isn't the solution for world hunger. It does, however, offer a compelling combination of popularity, ease of use and power. Software engineering is getting easier, and the Java platform is an important reason why. ☘

## About the Author

Rhett Guthrie is a Senior Technologist at ObjectSpace, Inc. working on ObjectSpace Voyager™, ObjectSpace's innovative distributed computing platform. Rhett can be reached at [rguthrie@objectspace.com](mailto:rguthrie@objectspace.com).

# Full Page Ad

**CALL FOR SUBSCRIPTIONS  
1 800 513-7111**International Subscriptions  
& Customer Service Inquiries**914 735-1900**

or by fax: 914 735-3922

E-Mail: [Subscribe@SYS-CON.com](mailto:Subscribe@SYS-CON.com)  
<http://www.SYS-CON.com>MAIL All Subscription Orders or  
Customer Service Inquiries to**JAVA DEVELOPER'S  
JOURNAL**

Java Developer's Journal

<http://www.JavaDevelopersJournal.com>**VRML DEVELOPER'S  
JOURNAL**

VRML Developer's Journal

[VRMLJournal.com](http://VRMLJournal.com)**NLC National JAVA  
LEARNING CENTER**

National Java Learning Center, Inc.

**JAVA DEVELOPER'S  
JOURNAL**  
1997 JAVA Products & Services  
**Buyer's Guide**  
& Internet Directory

JDJ Buyer's Guide

[JavaBuyersGuide.com](http://JavaBuyersGuide.com)**WEB-PRO  
DEVELOPER'S SUPPLEMENT**

Web-Pro Developer's Supplement

SYS-CON Publications, Inc.  
39 E. Central Ave.  
Pearl River, NY 10965 – USA**EDITORIAL OFFICES**

Phone: 914 735-1900

Fax: 914 735-3922

**ADVERTISING & SALES OFFICE**

Phone: 914 735-0300

Fax: 914 735-7302

**CUSTOMER SERVICE**

Phone: 914 735-1900

Fax: 914 735-3922

**DESIGN & PRODUCTION**

Phone: 914 735-7300

Fax: 914 735-6547

**DISTRIBUTED in the USA by  
International Periodical Distributors**674 Via De La Valle, Suite: 204  
Solana Beach, CA 92075  
Phone: 619 481-5928**Worldwide Distribution by  
Curtis Circulation Company**739 River Road,  
New Milford NJ 07646-3048  
Phone: 201 634-7400

David Gee



## The Component Choice for e-business

Have you heard the words 'build virtual teams, extend the corporation, manage the supply chain'? Are you convinced that e-business, enterprise applications deployed over the Web, Internet plus intranet plus extranet are the way to go? Chances are you've thought about this and your answer is yes. But what does that mean to you, right now, as we're one Web year into 1998?

As you and your development team move from pilot projects to implementation of enterprise-wide systems across the Internet, intranet and extranet, you'll need to consider and balance several critical elements. The right combination of component architecture, client/server tools, Internet practices and existing legacy systems is crucial. In terms of decisions to be made this year, I would like to address choosing the right component model as one of the most important issues you'll contend with. According to a recent report by Forrester Research, Inc., of companies interviewed, 44 percent have no object strategy now, but by the year 2000 only 4 percent foresee having no object strategy in place.

Components, by definition, are self-contained program modules that can interact with each other. The software community is creating components to speed application development and to build up a platform-independent base of reusable code. By incorporating a well-conceived component model, you can anticipate, drive and respond to changing market conditions, optimize reuse and create custom applications more quickly.

Since you're holding this magazine in your hands, you've already decided that Java is an important element of the e-business equation, whether you're beginning to experiment with Java or are an advanced Java programmer. In Java, almost everything is an object or component. The JavaBeans™ spec was written by JavaSoft in conjunction with numerous industry leaders, including IBM. JavaBeans is fast emerging as the portable, platform-neutral component model written in Java.

The JavaBeans component architecture is the ideal choice for developing network-aware applications that allow you to move within the enterprise or across the Internet. Unlike days past, you can't assume central control over deployment. If you anticipate deploying systems over a heterogeneous environment, you'll want new systems to connect and integrate with any other hardware or software that might be encountered on the Internet, intranet or extranet. The JavaBeans component model places no restrictions on where applications can

be deployed. And not coincidentally, JavaBeans connect into any other component model via bridges, including COM/DCOM. The opposite is not the case. The full COM environment – particularly Microsoft Transaction Server – will be available only on NT. Java and the JavaBeans architecture is the only model to consider with these goals in mind.

In 1997, we saw a lot of people building interesting client-side applications, just as before that we saw a plethora of spinning Java applets on the Web. The e-business equation rests on the belief that Java is not just a client-side model. Significant server-side Java initiatives are well under way, including IBM's massive San Francisco project. San Francisco has put over 300,000 lines of code in the hands of developers for creating run-your-business server applications. Over 250 companies have licensed the code so far, and the first of these applications will begin hitting the market later this year.

This server-side emphasis extends to the component model as well. The Enterprise JavaBeans spec is a component architecture for reusable server-side components to build business applications. IBM was a major contributor to this specification as well. Very soon we will begin to see support for scalable transactional application server components.

Other people in your organization who manage the desktops may be inclined to choose Microsoft's COM/DCOM model for your business. While COM/DCOM delivers decided benefits to the client, the real business benefit from component-based client/server lies in the business logic and applications that reside on the server. With leading visual development tools, Java's security model and built-in scalability, Enterprise JavaBeans is clearly the superior model.

Become the JavaBeans 'component proponent' within your company. You're going to be called on to make the quick changes and connections to the applications that run your e-business, so don't let the decision be made without you. Which component platform your organization adopts now will determine how and where you'll expand your business in the years ahead. ☛

### About the Author

As Program Director, alphaWorks & Java Marketing in IBM's Software Solutions, David Gee's role includes developing the company's Java marketing strategy, forging strategic business alliances and maintaining partner relationships with key industry influencers. You can learn more about IBM's Java initiatives at [www.ibm.com/java](http://www.ibm.com/java) and explore IBM's online research laboratory at [www.alphaWorks.ibm.com](http://www.alphaWorks.ibm.com).

# A Generalized Enumeration Mechanism *for Java*

*Help for  
developing  
complex  
iterators*

by Myung Ho Kim

**An iterator is a language mechanism that facilitates successive enumeration of all the elements of a collection in some definite order. Java provides an iterator-like interface called Enumeration.**



## Summary

An iterator is a language mechanism that facilitates successive enumeration of all the elements of a collection in some definite order. Java provides an iterator-like interface called Enumeration. The implementation model imposed by enumerations is known as cursor objects. It is not a simple task, however, to develop enumerations for non-linear data structures or for those with complex control in terms of cursors. This is because the enumeration procedure for a cursor is confined to use only simple control structures.

In this article, a new iterator mechanism is proposed for Java in which enumeration procedures can be described as if they were ordinary traversal procedures. It is fine to use arbitrary control structures such as recursive calls or infinite loops when developing enumeration procedures. Iterators are defined to implement the Enumeration interface and, thus, can be used in place of ordinary enumerations. The proposed iterator mechanism was implemented experimentally using threads and a zero-sized buffer.

## Introduction

An abstract data type should provide enough operations so that everything users need to do with its objects can be done without inspecting the implementation details. Since it is a common use of a collection to perform some action for its elements, we need a systematic way to access all elements. This method should be convenient to use and should not destruct the collection.

Some programming languages, such as CLU[1], Sather[2] and Icon[3], provide a mechanism called iterator (also known as enumerator) to handle this situation. In these languages, an iterator is invoked like a procedure but, instead of terminating with a result, it has many results which it produces one at a time. The produced items can be used in other parts that specify actions to be performed for each item.

Iterators are more important in object-oriented programming than they were in other paradigms because programmers routinely construct abstract data types for collections in this paradigm. The author of the data abstraction must provide an iterator because the representation of the objects of the type is not known to the user.

Java [4] provides an iterator-like interface called Enumeration (java.util.Enumeration). It is used by many classes in the standard Java libraries. In the case of the Vector class, the member that returns an enumeration is elements(). A typical loop using Enumeration to step through the elements of a Vector vec is as follows:

```
Enumeration elts = vec.elements();
while (elts.hasMoreElements())
    doSomething(elts.nextElement());
```

The Vector class, its related iterator class and the client program are interrelated, as illustrated in Figure 1.

The same technique can be applied to many other sorts of collections as well, if they provide members that return enumerations like the elements() member of the Vector class.

## Problems with Java Enumeration

The model Java provides to define iterators is known as cursors [5]. A cursor is an object that points into a collection and may be used to retrieve successive elements. The interface for cursors includes a test for completion (hasMoreElements()) and increment (nextElement()). The attributes of a cursor maintain the current state of the related iterator.

It is argued that iterators can be developed for all the common data structures using only cursors [6]. However, it is by no means a simple task to develop cursors for non-linear data structures like trees or graphs [7].

An iterator class for enumerating elements of binary trees will make the points clear (see Listing 1). Since the current node does not provide sufficient information to determine what the next node in an inorder sequence is, a stack is introduced to supplement the missing information.

This may be viewed as a non-recursive inorder traversal procedure tailored to meet the Enumeration interface. Even though it is possible, in principle, to remove all the recursions in any algorithm, the burden on the programmer is overwhelming when the algorithm is intertwined with complex control and/or data structures. What's worse is that the resulting program is far less readable than the original recursive one because control information for the enumeration procedure is dispersed at various places of the program.

The iterator construct of CLU is quite different from the Java Enumeration. A procedure-like module called an iter represents iterators. Since an iter module can be implemented using arbitrary algorithms, there is no need to explicitly maintain the current state to determine the next element. As a result, it is relatively easy to convert a traversal procedure into an iter module.

## Iterator as a Generalized Enumeration

We propose a new iterator mechanism for Java in which enumeration procedures can be described as if they were ordinary

traversal procedures. The whole interface of Java iterators is encapsulated in a single Java class called Iterator. From the user's point of view, an iterator is an instance of a class that extends the Iterator class which, in turn, implements the Enumeration interface.

Iterator developers should derive an iterator class from the Iterator class. The members hasMoreElements() and nextElement() are implemented already in the Iterator class and iterate() is the only member that should be implemented further.

iterate() corresponds to the iter module in CLU. It implements the enumeration procedure using arbitrary control structures including recursive calls and infinite loops. Any construct permissible within a plain member is also allowed. Additionally, it may contain many invocations of yieldElement() whose role is to enumerate an object and make it available to clients. Figure 2 illustrates the definition of the Iterator class.

The effect of calling an iterator is as follows. The first time an iterator is called via hasMoreElements() and nextElement(), it commences to execute iterate(). It does this until it reaches an invocation of yieldElement(), when it suspends execution and makes the value of expression visible to the corresponding call. Upon subsequent calls, it resumes execution just after yieldElement(), suspending whenever it reaches another yieldElement() until it exits.

An implementation of an iterator class for the inorder traversal of a binary tree using the Iterator mechanism is shown in Listing 2.

Even if the main use of iterators is to implement enumerations of data types, they are also useful in their own right. This is indicated by the next example (see Listing 3). Sort is a class for iterators that enu-

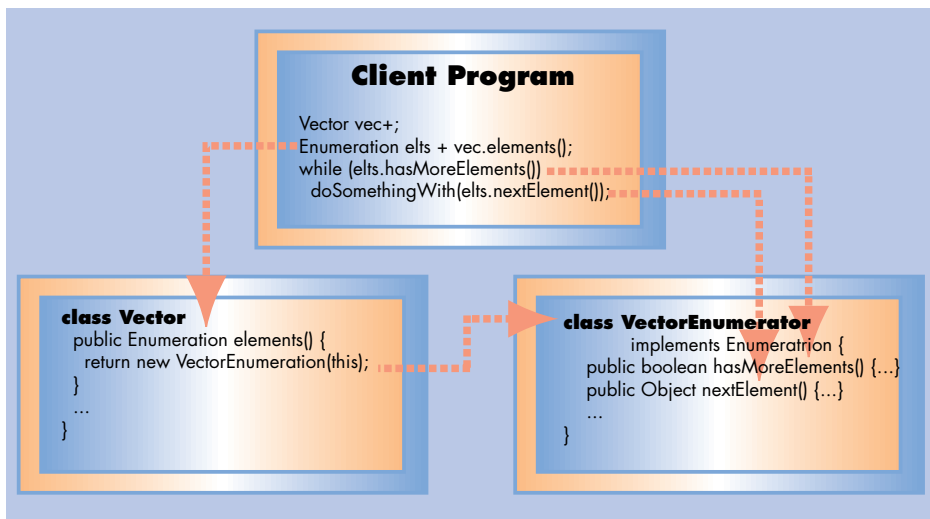


Figure 1: Vector, Enumeration and their client

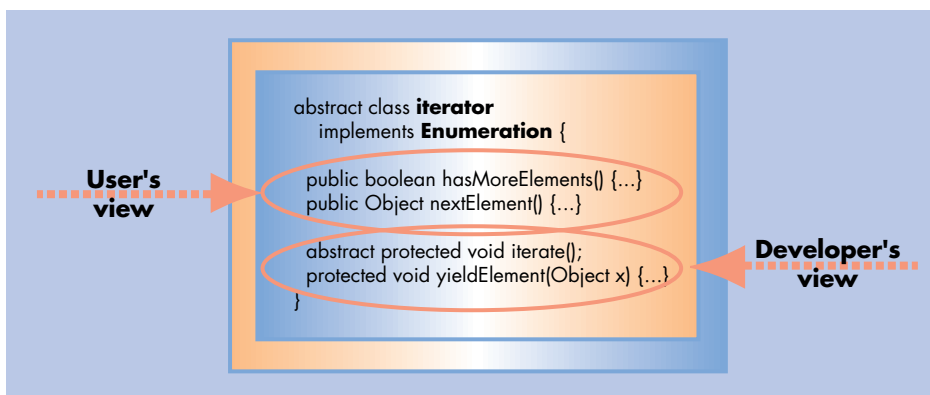


Figure 2: The Iterator class

merates elements of an array in ascending order using the quicksort algorithm [8]. Once again, the recursive nature of the original algorithm is well reflected by the enumeration procedure.

Our final example is a typical producer/consumer problem known as the Grune filter [9]. A filter accepts characters from an input stream and puts them out, replacing all occurrences of aa by b. Another filter replaces all occurrences of bb by c. Input is fed into the first filter, the output of which is fed into the second filter. What is actually observed is the output of the second filter. The filtering process can be programmed as in Listing 4.

It may seem quite simple at first, but programming an iterator class like Compact as a cursor is very difficult and error-prone. This is because it has many points of control that should be saved to determine what to enumerate next upon consecutive requests. In the Iterator model, there is no need to save control points explicitly. All that is needed is to invoke yieldElement() at various points.

Iterators are first-class objects and can be used in place of ordinary enumerations. This property exhibits maximum flexibility in programming with iterators [3]. Useful iterator-

related programming idioms (such as multiple iterators per loop or collection, parameterization of iterators to procedures or other iterators, binding iterators to variables and pipeline programming) can be applied.

The Grune filter is a typical example of pipeline programming. A pipeline is established when an iterator, acting like the consumer, embeds another iterator as a producer. In the following example, iterator AAB embeds an iterator for the standard input stream. By the same token, BBC embeds AAB.

```

Compact AAB = new Compact('a', 'b',
new StreamIterator
(new FileInputStream(
FileDescriptor.in)));
Compact BBC = new Compact('b', 'c', AAB);

```

It would be fair to say that programming techniques of this kind could also be applied to plain Java enumerations, but they are more strongly supported when iterators of arbitrary control can be developed easily.

## Implementation

An iterator (the enumeration procedure) and its clients, e.g. a procedure invoking

hasMoreElements() and nextElement() on that iterator, can be viewed as communicating sequential processes [10] that communicate by means of an intermediate buffer. Whenever a client requires another object, it takes one from the buffer. If the buffer is empty, it waits for the enumeration procedure to generate a new object and place it into the buffer. The enumeration procedure works to fill up the buffer and when it is full, it goes to sleep until the buffer is available again for new objects [11].

The buffer size chosen for the implementation of Java iterators is zero; thus, the enumeration procedure is activated directly by the clients upon every request for a new object. The buffer mechanism is implemented as a class named Buffer.

The basic idea of the implementation is as follows. An iterator and its client are executed in two different threads communicating through a buffer. The request for a new object is ultimately interpreted as an invocation of the get() member, which suspends the thread for the client until new data arrives at the buffer. The enumeration procedure takes control at this point, generating a new object and delivering it to the buffer by making a request for yieldElement(). It then invokes put(), which wakes up the thread for the client. Since there is neither true parallelism nor preemption between these two threads, they behave as if they were co-routines [12]. Figure 3 depicts the overall structure of the proposed implementation.

The thread for the iterator is responsible for starting the iterate() member that contains an enumeration procedure written by the iterator developer.

```

private class IteratorThread
extends Thread {
public void run()
{ iterate(); ...
}
}
private IteratorThread thread; ...

```

The constructor creates and starts a thread for the enumeration procedure. The thread is marked as a daemon thread to allow terminating the enumeration procedure when the client stops requests for further enumeration.

```

public Iterator() {
thread = new IteratorThread();
thread.setDaemon(true); thread.start();
}

```

HasMoreElements() should determine whether or not the enumeration procedure is still pending. This problem is harder to solve than it looks. It is insufficient to sim-

# Full Page Ad

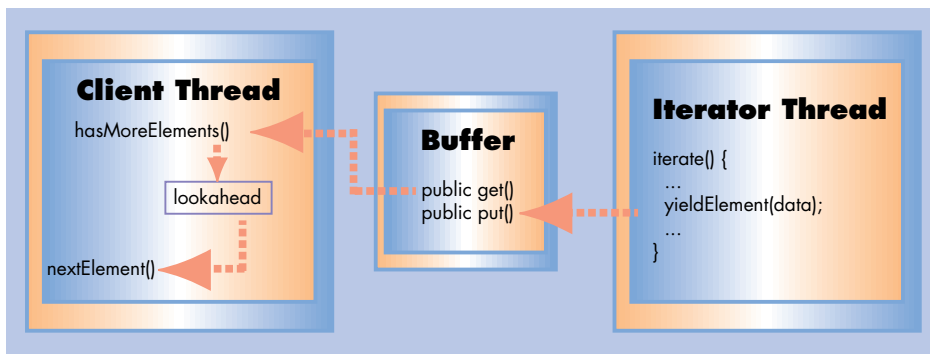


Figure 3: Implementation structure of iterators

ply check the liveness of the thread using `isAlive()` because there can be a race-condition between the two threads. The solution adopted here is pre-fetching an item delivered by the enumeration procedure. Pre-fetching is achieved by the `get()` request to the shared buffer.

```
private Buffer data = new Buffer();
```

The thread issuing that request is not woken up until some other thread (running the enumeration procedure) passes back the control. If the value delivered to the buffer is a meaningful item, it is a clue that the thread for the enumeration procedure is actually running and that the item stored in the buffer is one written by a `yieldElement()` call.

The value null was chosen for the representation of a meaningless item. The iterator thread delivers this value when there is a request for a new item even after the enumeration procedure is finished.

```
private class IteratorThread
    extends Thread {
    public void run() {
        iterate(); data.put(null); }
}
```

Lookahead is a temporary store for the pre-fetched item, which is shared between

`hasMoreElements()` and `nextElement()`.

```
private Object lookahead = null;
```

It should be a trivial task, by now, to determine whether the enumeration procedure is still pending. All that needs to be done is to check that the pre-fetched value is not null.

```
public final boolean hasMoreElements() {
    lookahead = data.get();
    return (lookahead != null);
}
```

`NextElement()` has only to return the value pre-fetched by the corresponding `hasMoreElements()` call.

```
public final Object nextElement() {
    return lookahead;
}
```

`YieldElement()` records the value to be enumerated next to the buffer which, in effect, passes the control back to the client.

```
protected final void yieldElement(Object x)
{
    data.put(x);
}
```

The implementation has a shortcoming in that every invocation of `nextElement()` should be preceded by exactly one corresponding invocation of `hasMoreElements()`.

There is also a small problem: the iterator thread tends to stay alive longer than it actually needs to when the iterator is used within a context that requires premature termination. This problem can be resolved if the thread could be terminated (stopped in the Java sense) explicitly by the programmer. A new member `stop()` is introduced to allow the required intervention.

```
abstract class Iterator implements Enumeration
{
    ...
    public final void stop() { ... }
}
```

The implementation of the `stop()` member is accomplished by terminating the iterator thread if it is still alive.

```
public final void stop() {
    if (thread.isAlive()) thread.stop();
}
```

Iterator threads that are neither finished nor stopped are maintained until the end of the whole program and then discarded when the main thread exits. Marking the iterator thread as a daemon already ensured this. The complete source code for both the `Iterator` and the `Buffer` is shown in Listing 5.

## Performance

The implementation has been tested using JDK 1.1.5[13] for SunOS 5.5.1. The results of the performance of cursors on the same workstation are shown and compared with the results of the proposed implementation. The benchmarks used are `Tree`, `Sort` and `Primes`. `Tree` is a sorting program based on binary search trees. `Sort` is also a sorting program based on the quick-sort algorithm. The `Primes` program enumerates prime numbers until the given count is reached.

Table 1 provides timings (in milliseconds) for example programs that created up to 500,000 random elements and consumed the same number of elements entirely. The “?” mark represents an `OutOfMemoryError` exception.

Cursors performed better than iterators in every test case. The overhead incurred during the synchronization of buffer operations was the dominant factor of the overall execution time when the enumeration procedures were very simple. When a large amount of computation was required for the enumeration of a sin-

Benchmarks	100	1,000	10,000	100,000	200,000	50,0000
Tree(c)	2	15	155	1,572	?(?)	?(?)
Tree(i)	11	120	1,281	11,959	?(?)	?(?)
Sort(c)	4	40	430	4,731	8,578	25,891
Sort(i)	12	120	1,231	12,698	26,102	65,084
Primes(c)	4	48	1043	26,309	70,963	280,387
Primes(i)	15	166	2,226	37,387	91,707	309,228

Table 1: Benchmark comparison of cursors (c) and iterators (i)

# Full Page Ad

## Resources

1. Liskov, B. et al., *CLU Reference Manual, Lecture Notes in Computer Science 114*, Springer-Verlag, 1981.
2. Murer S., Omohundro, S. and Szyperski, C., 'Sather Iterators: Object-Oriented Iteration Abstraction', Technical Report TR-93-045, ICSI, Berkeley, 1993.
3. Griswold, R.E. and Griswold, M.T., *The Icon Programming Language*, 2nd ed., Prentice-Hall, 1990.
4. Gosling J., Joy, B. and Steele, G., *The Java Language Specification*, Addison-Wesley, 1996.
5. Coplien, J., *Advanced C++ Programming Styles and Idioms*, Addison-Wesley, 1992.
6. Budd, T., *Multiparadigm Programming in Leda*, Addison-Wesley, 1995.
7. Horowitz, E., Sahni, S. and Mehta, D., *Fundamentals of Data Structures in C++*, Computer Science Press, 1995.
8. Hoare, C.A.R., 'Quicksort', *Comput. J.*, 5, (1), 10-15 (1962).
9. Grune, D., 'A View of Coroutines', *SIGPLAN Notices*, 12, (7), 75-81 (1977).
10. Hoare, C.A.R., *Communicating Sequential Processes*, Prentice-Hall International, 1985.
11. Baker, H.G., 'Iterators: Signs of Weakness in Object-Oriented Languages', *OOPS Messenger*, 4, (3), 18-25 (1993).
12. Marlin, C.D., *Coroutines: A Programming Methodology, a Language Design, and an Implementation*, Springer-Verlag, 1980.
13. JDK 1.1.5, available from <http://java.sun.com/>, JavaSoft, Sun Microsystems, Inc.

iterator developers can implement enumeration procedures as if they were ordinary traversal procedures. Arbitrary control structures, such as recursive calls or infinite loops, can also be used inside enumeration procedures. This level of expressive power is essential when writing iterators for non-linear data structures or for those with complex control.

Iterators developed using the proposed mechanism are first-class objects and can be used in whichever context an Enumeration makes sense. This property exhibits maximum flexibility in programming with iterators. The implementation is still a prototype but it faithfully realizes all the essential features of the proposed mechanism. ☛

### About the Author

Myung Ho Kim is an associate professor of Dong-A University, Pusan, Rep. of Korea. He has been teaching programming paradigms to undergraduate and graduate students for more than 10 years. Currently he is leading a project for developing a full-fledged interpreter and transformation system of a functional language in Java. He can be reached at: [mhkim@esther.donga.ac.kr](mailto:mhkim@esther.donga.ac.kr).



[mhkim@esther.donga.ac.kr](mailto:mhkim@esther.donga.ac.kr)

gle data item, however, iterators could run nearly as fast as cursors, which can be observed near the lower right corner of the table. Even if cursors are desirable for efficiency reasons, it would still be a useful programming strategy to use iterators during the initial development stages and

then translate them into cursors later.

## Conclusion

A new mechanism for the Java programming language is presented in this article that aids programmers in developing complex iterators. In the proposed mechanism,

### Listing 1.

```
class Tree {
    class TreeNode { ... }
    public Enumeration elements() {
        return new InorderIterator(rootNode);
    }
    class InorderIterator implements Enumeration {
        public InorderIterator(TreeNode node) {
            currentNode = node; walk = new Stack();
        }
        public boolean hasMoreElements() {
            while (currentNode != null) {
                walk.push(currentNode);
                currentNode = currentNode.leftChild;
            }
            return walk.empty();
        }
        public Object nextElement() {
            currentNode = (TreeNode)walk.pop();
            Object value = currentNode.data;
            currentNode = currentNode.rightChild;
            return value;
        }
        private TreeNode currentNode;
        private Stack walk;
    }
    private TreeNode rootNode;
}
```

### Listing 2.

```
class InorderIterator extends Iterator {
    public InorderIterator(TreeNode node) {
        rootNode = node;
    }
    protected void iterate() { inorder(rootNode); }
    private void inorder(TreeNode currentNode) {
        if (currentNode != null) {
            inorder(currentNode.leftChild);
```

```
yieldElement(currentNode.data);
            inorder(currentNode.rightChild);
        }
    }
    private TreeNode rootNode;
}
```

### Listing 3.

```
class Sort extends Iterator {
    public Sort(double[] v, int low, int high) {
        data = v; left = low; right = high + 1;
    }
    protected void iterate() { quick(left, right); }
    private void quick(int left, int right) {
        if (left <= right) {
            int pos = partition(left, right + 1);
            quick(left, pos - 1);
            yieldElement(new Integer(data[pos]));
            quick(pos + 1, right);
        }
    }
    private int partition(int low, int high) { ... }
    private double[] data;
    private int left, right;
}
```

### Listing 4.

```
class Compact extends Iterator {
    public Compact(char c1, char c2, Enumeration enum) {
        s1 = new Character(c1); s2 = new Character(c2);
        theEnum = enum;
    }
    protected void iterate() {
        while (theEnum.hasMoreElements()) {
            Object s = theEnum.nextElement();
            if (s.equals(s1)) {
                if (theEnum.hasMoreElements()) {
                    s = theEnum.nextElement();
```

```

        if (s.equals(s1))
            s = s2;
        else
            yieldElement(s1);
    }
    else {
        yieldElement(s1);
        break;
    }
}
yieldElement(s);
}
}
private Enumeration theEnum;
private Character s1, s2;
}

```

### **Listing 5: Complete Source Codes for Iterator API.**

```

// class Iterator
abstract class Iterator implements java.util.Enumeration {
    public Iterator() {
        thread = new IteratorThread();
        thread.setDaemon(true); thread.start();
    }
    public final boolean hasNextElements() {
        lookahead = data.get();
        return (lookahead != null);
    }
    public final Object nextElement() {
        return lookahead;
    }
    protected final void yieldElement(Object x) {
        data.put(x);
    }
    public final void stop() {
        if (thread.isAlive()) thread.stop();
    }
}

```

```

abstract protected void iterate();
private class IteratorThread extends Thread {
    public void run() {
        iterate(); data.put(null);
    }
}

private Token data = new Buffer();
private IteratorThread thread;
private Object lookahead = null;
}

// class Buffer
final class Buffer {
    public synchronized Object get() {
        requestIssued = true;
        notify();
        while (! dataAvailable)
            try { wait(); }
            catch (InterruptedException ex) {}
        dataAvailable = false;
        return (data);
    }
    public synchronized void put(Object item) {
        while (! requestIssued)
            try { wait(); }
            catch (InterruptedException ex) {}
        requestIssued = false;
        data = item;
        dataAvailable = true;
        notify();
    }

    private Object data;
    private boolean dataAvailable = false;
    private boolean requestIssued = false;
}

```

# 1/2 Ad

Java Security

# PROTECTION DOMAINS

## New Security Features in JDK 1.2

by Jahan Moreh

Many existing security models burden programmers with the obligation to understand, code and enforce the security policy of an organization. The current state-of-the-art model for implementing security suggests that a developer should not directly be involved in implementing an organization's security policy. New security features in JDK 1.2 help realize this model.





Java is rapidly evolving from just a useful language for developing Web-based applets to an enterprise platform for developing and deploying mission critical applications. An enterprise-class application must possess many characteristics. Comprehensive security is, inarguably, one of these characteristics.

Many existing security models burden programmers with the obligation to understand, code and enforce the security policy of an organization. The current state-of-the-art model for implementing security suggests that a developer should not directly be involved in implementing an organization's security policy. This model provides two important advantages:

1. It reduces the likelihood of an incorrect implementation, thus increasing the security of the overall system.
2. It allows dynamic changes to the security policy to take effect quickly. Typically, changes to an organization's security would not require an application recompile. Ideally, changes to the organization's security policy will not even require application rerun/restart.

From its inception, Java has had several built-in security features. The original JDK 1.0.x provided security through three mechanisms:

1. The Java language code safety via the sandbox model
2. The Java built-in bytecode verifier
3. The Java SecurityManager and ClassLoader classes

JDK 1.1.x provided further security by introducing the concept of trusted applets. JDK 1.1 also provided classes for producing and verifying digital signatures of arbitrary data. Finally, the Java Cryptographic Extension (JCE) to JDK1.1 introduced the notion of privacy-protection of arbitrary data through encryption classes.

*JDJ* has published a number of articles dealing with Java security. Specifically, in an article entitled *Implementing a Security Policy* (*JDJ* Vol. 2, Issue 8), Qusay Mahmoud wrote on practical uses of the Java SecurityManager class. Then, in an article entitled *Java Security: Beyond Code Safety* (*JDJ* Vol 2, Issue 12), I wrote about practical uses of the cryptographic interfaces of JDK 1.1, including their use in creating and verifying the source of trusted applets.

This article focuses on Protection Domains – a new JDK 1.2 security feature for implementing fine grain access control.

### Trust Model before JDK 1.2

The original JDK implemented a simple and somewhat effective access control model. This model – known to Java devel-

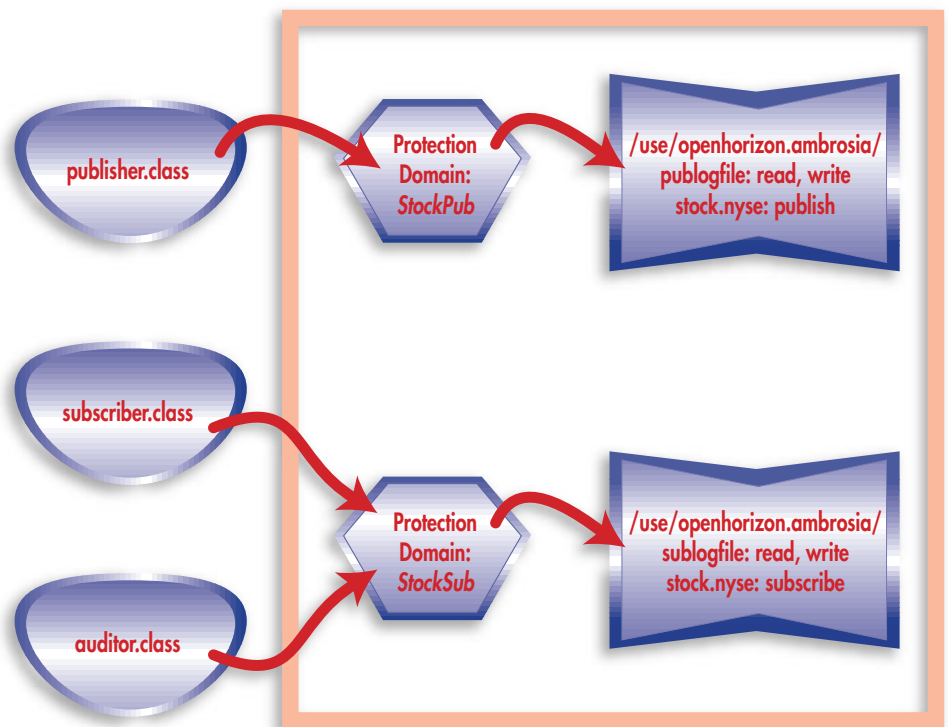


Figure 1: Classes, domains and permissions

opers as the sandbox – provides a binary choice within the runtime environment: trust everything that is local but do not trust anything that is downloaded. In other words, the Java runtime completely trusts every Java application and local applet, and it completely restricts every downloaded Java applet. However, most applets need to interact with users in a personalized way. This means that in order for an applet to provide a useful function – especially in the context of electronic business – it does need access to some local resources. Simultaneously, the security policy of most organizations demands that local applications not be given open access to every resource. Therefore, the original JDK's trust model is too restrictive on applets and too permissive on local applications.

JDK 1.1 expanded the sandbox model via signed, trusted applets. Basically, the client who downloads the applet can choose to trust the originator. Assuming that the digital signature of the downloaded applet verifies correctly, the applet would have access to local resources (files, network connections, etc.). Therefore, a client can treat each applet differently based on its digital signature. However, the access control model for a given applet remains binary. If trusted, it has access to all resources and if not, it must run within the sandbox.

JDK 1.2 further expands the binary access control model of JDK1.1 with fine grain access control. JDK 1.2 introduces the concept of *protection domains* which

allows a client to specify exactly which resources a given applet or application may access and for what reason (e.g., read a file, connect to a host, etc.). Therefore, JDK 1.2 allows for the correct implementation of two important and complementary security elements:

1. Allowing a downloaded applet access to predefined local resources. That is, opening the sandbox as determined by the local security policy.
2. Denying a local application access to predefined local resources. That is, restricting local applications from having access to all resources.

### Protection Domains

A protection domain is a set of classes currently accessible by a principal. Interestingly enough, the original sandbox model represents a protection domain with a fixed, static boundary. Under the JDK 1.2 model, a security administrator can dynamically change the boundaries of a protection domain using permissions. Here's how it works. Each class belongs to one, and only one, protection domain. Associated with each domain is a set of permissions, reflecting the security policy of the organization. Figure 1 illustrates the relationship between classes, protection domains and permissions. As shown, the class publisher belongs to a protection domain called StockPub. Classes belonging to this domain can read and write a file called /usr/openhorizon/ambrosia/publogfile. Additionally, classes belonging to this domain have publish permission on a



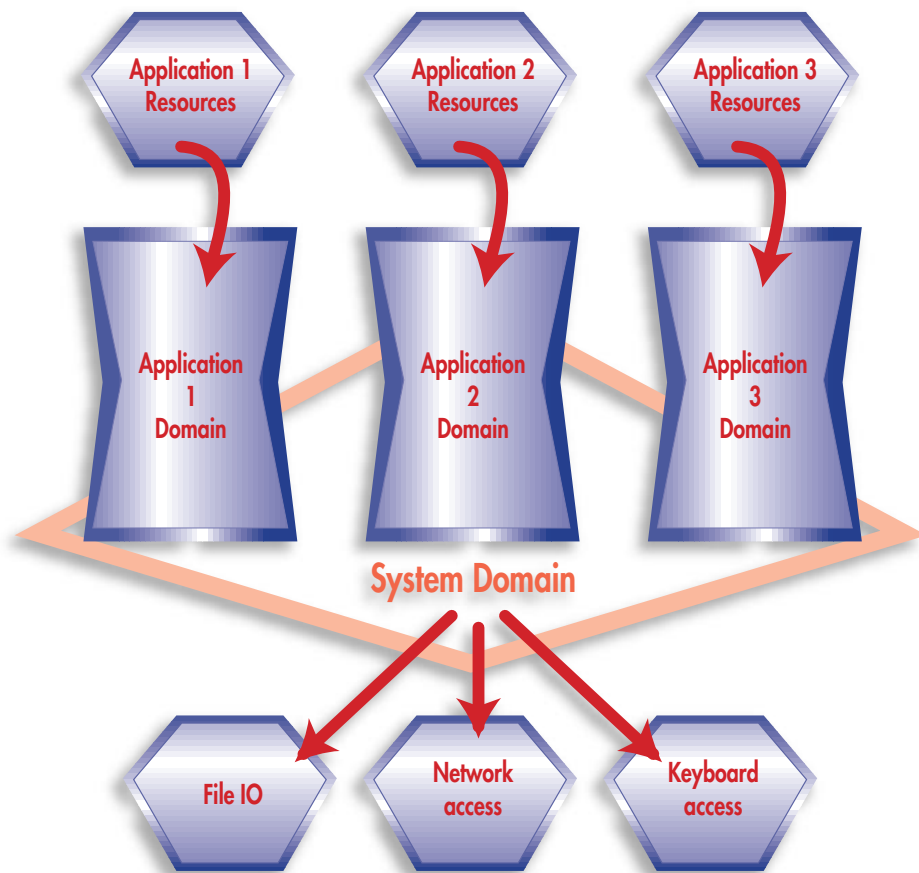


Figure 2: Application and system domains

resource called `stock.nyse`. Subscriber and auditor classes belong to a protection domain designated by `StockSub`. Classes belonging to this domain can read and write a file called `/usr/openhorizon/ambrosia/sublogfile`. Additionally, these classes have subscribe permission to a resource called `stock.nyse`. **Note:** Even though a class cannot belong to more than one protection domain, a protection domain can be shared by multiple classes. Thus, a security administrator can logically group classes and assign them to a protection domain.

### Enforcing Access Control

So, who enforces the actual access control? It depends. The Java runtime environment controls external resources such as the file system, network connections, the keyboard and the mouse. It is the Java runtime environment that enforces access control to these resources. In the example illustrated in Figure 1, it is the Java runtime environment that decides `auditor.class` can read and write `/usr/openhorizon/sublogfile`, and that this class cannot delete the same file. This is because the administrator has explicitly granted read and write permission to the file but not delete permission. On the other hand, the Java runtime environment has no way to interpret

or enforce application dependent resources. Again, in Figure 1, `publish` and `subscribe` permissions, and the resource represented by `stock.nyse`, are completely application dependent. (These operations are pertinent to an application built on top of a `publish/subscribe` middleware. The resource `stock.nyse` represents a subject for publishing information and subscribing to information.) Thus, each application must enforce access to its resources in the appropriate manner. For this reason, JDK 1.2 classifies resources according to two general categories: 1: system resources fall into the system domain; and, 2: application resources fall into the application domain. Figure 2 illustrates this classification.

### Crossing Protection Domains

We mentioned earlier that a class belongs to one and only one protection domain. Simultaneously, an application may need access to resources that belong to multiple domains. For example, in Figure 1, the file `/usr/openhorizon/publogfile` belongs to the system domain while the resource `stock.nyse` belongs to the application domain. It turns out that a single thread of execution usually traverses more than one class and, hence, it traverses more than one protection domain. For example, if a client downloads the applet

`publisher.class`, it is likely that this class will publish a message and write to a log file. As a thread traverses multiple domains, it is crucial that it does not become more privileged. For example, a thread that originates in `publisher.class` may invoke a method from a class in the `java.io` package to write to a file. The write operation must occur in the system domain as it controls access to the file resource. However, `publisher.class` must not gain additional privileges (e.g., the ability to attempt to write to any file) as it enters the system domain. Conversely, a thread originating in the system domain may invoke a method in the application domain. For example, an AWT thread may call the applet's `paint` method. Again, it is crucial that the thread has its privilege reduced to match the permissions of the specific application domain. Luckily, the Java runtime environment correctly enforces this crucial security semantic. Thus, an application belonging to a less powerful domain cannot gain additional privileges as a result of invoking a method in a more powerful domain. Simultaneously, a thread originating from a more powerful domain loses some of its privileges when it calls a method in a less powerful domain. We can generalize this by stating that the permission of an execution thread is the intersection of the permissions granted to all protection domains traversed by that thread.

### Steps Involved in Creating a Security Policy

In order to create an effective access control security policy, you must follow these steps:

1. Determine the principals to whom you wish to grant certain permissions. Obtain a digital certificate for each principal. This is a one time effort that aids in verifying the origin of applets.
2. If applicable, determine the URL codebase when the applet must originate. You may wish to grant certain permissions to an applet that is digitally signed by a principal but only if it originates from a specific URL.
3. Define the set of resources for which you want to grant permissions. These could be system resources (e.g. files, network connections, etc.) or application specific resources.
4. Define the exact set of permissions you would like to grant.

After you complete these tasks, you are ready to create a security policy file. A security policy file has a fairly straightforward format. As illustrated in Listing 1, each entry in the policy file begins with the

Full pg

reserved word “grant.” Then, within each entry, the reserved word “permission” delimits the set of permissions granted.

Listing 2 shows progressively elaborate access control entries in a policy file. Lines 1-3 designate that any applet, originated from anywhere, can read and write the file /usr/tmp/logfile. This is because the grant line does not specify a SignedBy clause or a CodeBase clause. **Note:** java.io.FilePermission is a built-in permission in JDK 1.2; it must be specified using its fully qualified package name.

In Listing 2 (lines 5-8), we specify permissions for an applet that is digitally signed by a principal called “openhorizon.” As shown, such an applet would be able to connect to a socket on the host “www.openhorizon.com” as long as the listening port on the host is in the range of 8000-8200. Again, note that java.net.SocketPermission is a built-in permission in JDK1.2 and must be specified using its fully qualified package name.

In Listing 2 (lines 10-13), we specify permission for an applet that is digitally signed by the principal “openhorizon” and originates from www.openhorizon.com/Ambrosia/demo. As shown, such an applet would be able to connect to port 8506 on the host www.openhorizon.com. Additionally, the applet would have publish permission on the resource demo.nyse.stock. Note that com.openhorizon.client.PublishPermission is a custom permission defined by the package of the downloaded applet. Moreover, demo.nyse.stock is a resource defined by the package of the downloaded applet. You may have noticed that unlike the file/read and socket/connect examples, PublishPermission does not have a specific action. This is perfectly reasonable in the JDK 1.2 model; if a permission is granular enough, an action need not be specified.

Finally, Listing 2 (lines 15-19) shows a very elaborate entry in the policy file. Here, we grant certain permissions to applets that originate from the specified URL and are signed by “openhorizon.” Using the signedBy clause in the permission entry (i.e., signedBy “OHISecurityOfficer”), we direct the Java runtime to verify the digital signature of the bytecode that implements the permission com.openhorizon.client.PublishPermission. This feature allows for a very powerful access control mechanism: it prevents permission spoofing.

## Permission Classes in JDK 1.2

Three classes in JDK 1.2 form the basis of permissions:

1. java.security.Permission is an abstract class. Its subclasses represent specific

*“Based on the local security policy, JDK 1.2 can open the sandbox for downloaded applets or restrict access by local applets/applications”*

permissions (e.g., file access permission, network connect permission, etc.).

2. java.security.PermissionCollection is an aggregate of homogenous permissions. It is useful for grouping similar permissions and granting all permissions as a group. For example, one can create a FileAccess permission collection that represents read, write and delete access to certain files. FileAccess would be a PermissionCollection that comprises three java.io.FilePermission classes: read, write and delete.
3. java.security.Permissions holds a heterogeneous collection of permissions. In other words, if you wanted to group several different types of permission collections and grant them as a group, you would use this class. For example, you may define a TotalAccess permission that comprises FileAccess and NetAccess permission collections.

JDK 1.2 comes with a number of built-in permission classes. Figure 2 illustrates the three base classes and their relationship to built-in derived classes. Two important built-in permissions are explained here:

1. **java.io.FilePermission:** This class is a subclass of java.security.Permission. It controls access to files and directories. This class allows wildcard specification for files. Listing 3 shows some examples of file permissions. As shown, P1 signifies a permission to read and write the specified file. P2 represents a permission to delete all files under the subtree /usr/ambrosia/log. P3 signifies a permission to execute any program under the directory /usr/ambrosia/bin. Finally, P4 represents a permission to read all files in the file system.

2. **java.net.SocketPermission:** This class is a subclass of java.security.permission. It controls access to network resources. This class also provides for wildcard specification for host names and ports. Listing 3 shows some examples of network permissions. As shown, P5 signifies a permission to listen to port 8506 on the host demo.openhorizon.com. P6 represents a permission to connect to any host in the com domain.

## Enforcing Application-Dependent Access Control

We mentioned earlier that the Java runtime environment enforces access to files, network resources, the keyboard, the mouse, etc. We also mentioned that the Java runtime environment has no way to interpret or enforce application dependent resources. Therefore, it is the application that must enforce access control to these type of resources. Fortunately, this is really easy. JDK 1.2 defines a class called AccessController. This class has a very important static method called checkPermission. All an application programmer needs to do is invoke this method and pass it the permission that needs to be checked. If the thread of execution has the given permission, as specified by the security policy file, then AccessController.checkPermission returns quietly. Otherwise, this method throws an exception.

## Conclusion

Java is rapidly evolving to a mature platform for building mission critical applications. Comprehensive security is an important element of mission critical applications. With the introduction of Protection Domains in JDK 1.2, Java developers have the means to implement an effective security policy for fine grain access control. Together with Java Cryptographic Architecture (JCA) and Java Cryptographic Extension (JCE), Protection Domains form the basis for building a comprehensive security package into the Java Development Kit. ☛

### About the Author

Jahan Moreh is a distributed system architect with Michigan Group, Inc. He specializes in middleware architecture and information security. You can reach him via e-mail at [jmoreh@michigangrp.com](mailto:jmoreh@michigangrp.com).



[jmoreh@michigangrp.com](mailto:jmoreh@michigangrp.com)

**Don't Type it... Download it!**  
Access the source code for this and other articles appearing in this issue at [JavaDevelopersJournal.com](http://www.JavaDevelopersJournal.com)

Ad

# Reflection & Introspection: Objects Exposed

*Techniques for Dynamically Viewing Java Classes*

by Ajit Sagar & Israel Hilerio

One of the salient aspects of the Java language is the control it gives to developers for dynamically generating and reusing code. This allows the language to offer Java programmers the ability to write code in which the actual behavior is determined at runtime. Of the eleven buzzwords used to define Java, this article is going to focus on the dynamic nature of the Java programming language.

Java achieves its 'dynamism' through the use of its `ClassLoader` and two core APIs: Reflection and Introspection. We will begin this article with an introduction to these APIs and their roles in Java-based software development. The APIs are compared in the light of Bean-based component introspection and user-defined component reflection. This is followed by brief descriptions of each of the APIs. We will then walk the reader through a series of examples

that illustrate the development of some programming utilities that may be used by readers in Java application development. It is assumed that readers have some familiarity with the JDK 1.1.x APIs. We will not attempt to cover the APIs in detail.

This article is the first in a series of two. A subsequent article will lead the readers through the development of a new category of dynamically generated adapters called Dynamic Adapters. This article concludes with a very brief introduction to the Adapter design pattern and Dynamic Adapters.

## The Role of Reflection and Introspection in Java Development

Reflection and introspection are programming facilities in the Java programming language that allow an object to discover information about itself and other

objects at runtime.

Webster's dictionary defines reflection as "the act of giving back or showing an image of." Reflection in Java allows the developer to create objects that can:

- Construct new class instances and new arrays
- Access and modify fields of objects and classes
- Invoke methods on objects and classes
- Access and modify the elements of arrays

So, how is that different from regular object-oriented programming? Objects defined in other object-oriented programming languages can accomplish any of the



above. However, in Java, the use of the Reflection API allows an object to do the above on another object (or on itself) without knowing at compile time what the object/class being acted upon looks like.

The state and behavior of the object/class can be determined at runtime. Figure 1 illustrates the role of reflection in Java programming.

Webster's dictionary defines

## Introspection Uses Reflection

Reflection and introspection are very closely related. Reflection is a low-level facility that allows the code to examine the internals of any class or object at runtime. Introspection builds on this facility and provides a more convenient interface for examining Beans. In fact, the relationship between reflection and introspection is very similar to the relationship between JavaBeans and other Java classes. JavaBeans are simply normal Java objects with certain design patterns enforced in their nomenclature. Introspection assumes these design patterns on the object that it is inspecting and uses low-level reflection to examine the object's internals.

## The Reflection API

The Reflection API became a part of core Java with release 1.1 of the JDK. The API is defined across the following:

- The new methods added to the `java.lang.Class` class in JDK 1.1
- The `java.lang.reflect` package defined in JDK 1.1

The class `java.lang.Class` contains methods that return instances of classes/interfaces defined in the `java.lang.reflect` package. A detailed description of the API is beyond the scope of this article and can be found in any standard Java text. However, the classes that comprise the Reflection API are listed in Table 1.

## The Introspection API

The Introspection API consists of several classes in the `java.beans` package. Again, a detailed description of the API is beyond the scope of this article and can be found in any standard Java text. The main classes in the Introspection API are listed in Table 2.

## The Costs of Usage

Reflection and Introspection are powerful tools that contribute to the flexibility provided by the Java language. However, these APIs should be used only as needed and after taking into account the costs associated with their usage:

- Reflection and Introspection method calls have a substantial performance overhead.
- Using reflection makes the code much more complex and harder to understand than using direct method calls.
- Errors in method invocation are discovered at runtime instead of being caught by the compiler.
- The code becomes type-unsafe.

The Reflection and Introspection APIs should be used only when other forms of

object-oriented programming are not appropriate.

The following examples demonstrate the use of Reflection and Introspection to develop some useful Java utilities.

## Cookie Factory

Our first example illustrates the use of Reflection to build a utility that allows us to instantiate objects of types derived from a "Cookie" interface. The actual type of the object instantiated is determined by a String parameter, which contains the name of the actual class. The code for the example is shown in Listings 1 and 2.

Listing 1 defines the Cookie interface and the derived classes. The Cookie interface is simply a marker interface which is implemented by the classes `FortuneCookie` and `MisFortuneCookie`. Both these classes define a single static method which prints out a string and returns a new instance of the respective class.

Listing 2 shows the `CookieFactory` class which is capable of producing objects derived from the "Cookie" interface. It defines a single method `createCookie` that takes a String parameter, `className`. The Class corresponding to this name is obtained from the `Class` class by calling

```
c = Class.forName(className);
```

Once we have the class, we need to obtain the method to be called on it. The name of the method is "newCookie." In this example, we are assuming that the name of the method is available at this point. The parameter types for the method are filled in an array of type `Class` and this is used to get the actual Method object as follows:

```
method = c.getMethod("newCookie", pTypes);
```

Once the Method object is available, the static method is invoked on the class after constructing an array of Objects that contains the actual parameter instances:

```
cookie = (Cookie)method.invoke(c, params);
```

A simple tester for the class is provided in the `main()` method. This first constructs the `CookieFactory` and then creates instances of the `FortuneCookie` and `MisFortuneCookie` class. The output from the program is shown in Figure 3.

## An X-Ray Class

Our second example illustrates the use of reflection to build a utility that allows us to view all the methods, constructors, fields, interfaces and inheritance for a supplied class. The class being X-rayed is specified by a String parameter which contains

introspection as "observation or examination of one's own state." In

Java, introspection is used in the context of JavaBeans, which define Java's component model. Introspection is used to allow a Bean to discover the properties, methods and events of another Bean at runtime. This enables developers to design and build their own Beans without knowing about the internals of another Bean.

Introspection is used by visual builder tools to introspect on Beans; i.e., to determine what properties are exposed by the Bean, what public methods it provides and what events it can generate. However, Introspection is a facility that is available to all Java classes, not just JavaBeans. Figure 2 illustrates the role of introspection in Java programming.

## java.lang package

Class/Interface	Description
Class	A class that represents a Java class or interface. Provides all the information about the class such as the constructors, fields, superclass, methods, modifiers, interfaces, declaring class, etc.; as well as the capability to reflect into the inner classes contained in a class.

## java.lang.reflect package

Class/Interface	Description
Array	A class that contains methods that allow getting or setting the values in an array, determine the length of the array and create new instances of arrays.
Constructor	A class that represents a constructor method of a class. Instances of Constructor are obtained by calling <code>getConstructor()</code> and related methods of <code>java.lang.Class</code>
Field	A class that represents a field of a class. Instances of Constructor are obtained by calling <code>getField()</code> and related methods of <code>java.lang.Class</code>
Member	An interface that defines the methods shared by all members (fields, methods, and constructors) of a class.
Method	A class that represents a Method. Instances of Method are obtained by calling <code>getMethod()</code> and related methods of <code>java.lang.Class</code>
Modifier	A class that defines a number of constants and static methods that are used to interpret modifiers like <code>public</code> , <code>abstract</code> , <code>final</code> , etc.

Table 1: The Reflection API Classes

## java.beans package

Class/Interface	Description
BeanDescriptor	A class that describes global information relative to a Bean.
BeanInfo	An interface which is implemented to provide detailed information about each and every Bean property via the descriptor classes.
FeatureDescriptor	A base class for Bean introspection of features (methods, properties, event sets, and parameters).
EventSetDescriptor	A class that describes the event sets associated with the Bean.
MethodDescriptor	A class that describes a particular method published by the Bean.
PropertyDescriptor/ IndexedProperty Descriptor	Classes that describe a particular property published by the Bean.
Introspector	A class that uses implicit (specified by the developer in the <code>BeanInfo</code> class) and explicit (using automatic inspection of beans design patterns and low-level reflection) information to build a <code>BeanInfo</code> object that completely describes the Bean at design-time. Used by Bean-aware tools to learn about the Bean.
SimpleBeanInfo	Provides a basic framework for a custom <code>BeanInfo</code> class.

Table 2: The Introspection API Classes

the name of the actual class. The code for the example is shown in Listing 4.

In order for the X-ray class program to determine the methods, constructors, fields and interfaces contained in the requested class, it must instantiate an object of the class by calling:

```
c = Class.forName(className);
```

After instantiating the class, the utility determines the selected operation on that class based on a second user-supplied string parameter, which can have one of the following values:

*localMethods*  
*allMethods*  
*Constructors*  
*fields*  
*interface*  
*inheritance*

The local methods contained in the specified class may be found by calling:

```
methodList = c.getDeclaredMethods();
```

This call returns a `Method[]` that contains all the methods declared in the local class including private, protected and public. This call excludes inherited methods.

A list of all the public methods, both inherited and local, may be obtained by calling:

```
methodList = c.getMethods();
```

Constructor methods are not included in the return `Method[]` of this call. Retrieving a list of constructors for a specific class

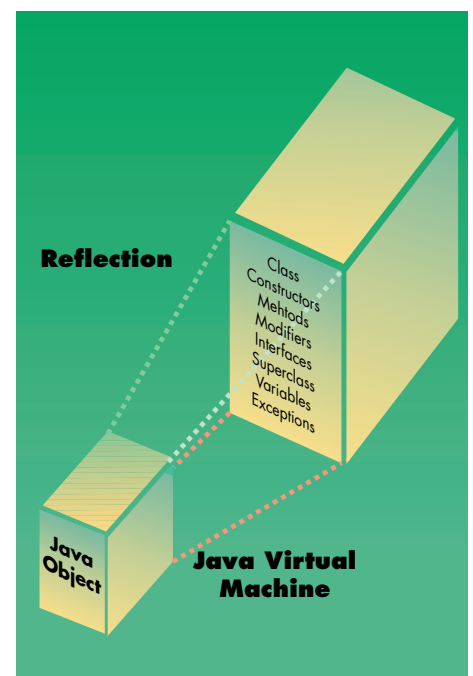


Figure 1: Role of reflection in Java programming



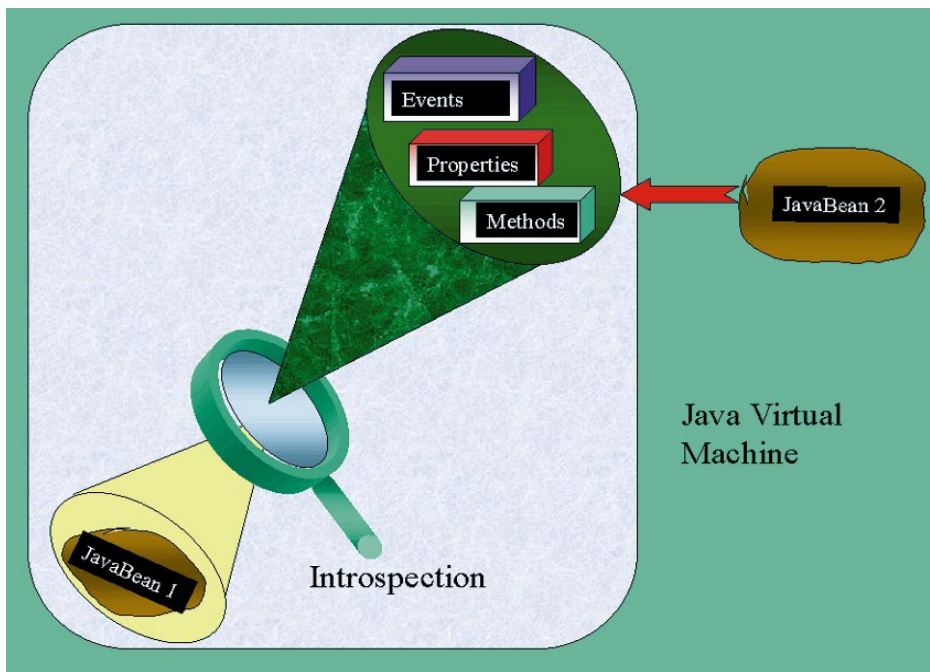


Figure 2: Role of introspection in Java programming

the superclasses in the inheritance tree. This is done in a while loop by calling:

```
classRef = c.getSuperclass();
```

We use this mechanism to return a class[] with all of the classes that participate in the extension of the local class. The output of the program for obtaining the inheritance hierarchy of the java.applet.Applet class is given in Figure 4.

Notice that of the various methods presented on this section, the only method that recursively provided information contained in its inheritance tree was the c.getMethods() call. The other methods only provided information contained by the local class.

### An X-Ray Bean

Our third example illustrates the use of Introspection to build a utility that allows us to view all the methods, properties and events for a supplied JavaBean class. The Bean being X-rayed is specified by a String parameter which contains the name of the actual Bean class. The code for the example is shown in Listing 3.

In order for the X-ray Bean program to determine the methods, properties and events contained in the requested class, it must instantiate an object of the class by calling:

```
c = Class.forName(className);
```

After instantiating the class, the utility must access the BeanInfo for the instantiated Bean. BeanInfo data can be accessed via the Introspector class by calling:

```
bi = Introspector.getBeanInfo(c);
```

Next, the utility determines the selected operation on that class based on the second user-supplied parameter entered at the command line (i.e., methods, properties and events). The localMethods contained by the specified Bean can be found by calling:

```
methodDescriptorList = bi.getMethodDescriptors();
```

This call returns a MethodDescriptor[] that contains a description of all of the methods contained by this Bean. The type of method descriptor returned by this call contains a complete list of all the public methods contained within the inheritance tree of this Bean. In order to access the actual method instances, we need to iterate through the methodDescriptorList and obtain the method by calling:

```
methodRef = methodDescriptorList[i].getMethod();
```

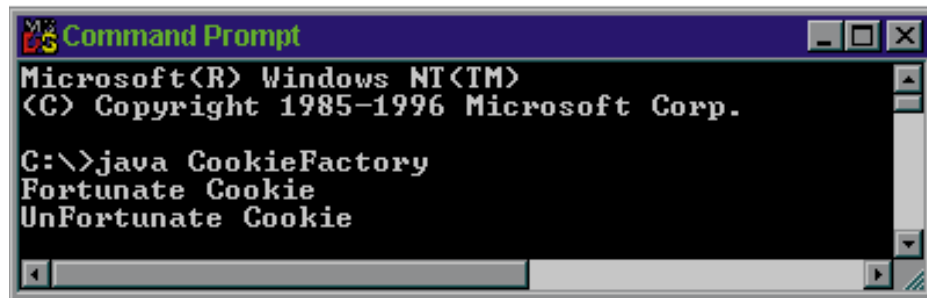


Figure 3

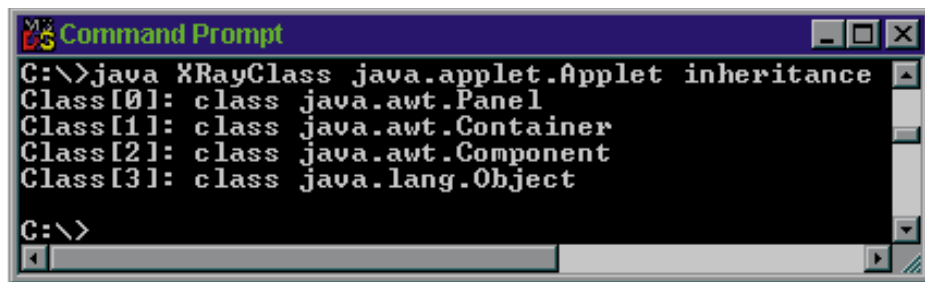


Figure 4

can be accomplished by calling:

```
constructorList = c.getDeclaredConstructors();
```

This call returns a constructor[] that contains all of the private, protected and public constructors declared on the local class. A list that includes only the public constructors can be constructed by calling:

```
constructorList = c.getConstructors();
```

Class variables can be retrieved as Field[] information. To access a complete list of fields from a class including private, protected and public, we call:

```
fieldList = c.getDeclaredFields();
```

A list that includes only the public fields can be retrieved by calling:

```
fieldList = c.getFields();
```

Information concerning the interfaces implemented by a class can be accessed by calling:

```
interfaceList = c.getInterfaces();
```

This call returns a Class[] that contains all of the interfaces implemented by the local class. Notice that this call doesn't have a getDeclaredInterfaces() counterpart like the other methods.

To access the inheritance information in the class, we get the name of each one of

From each one of these values, we are able to build a Method[] list that can be displayed by the utility. This includes no constructor method information. To access the Bean constructor information, you must use reflection.

Bean properties can be retrieved via PropertyDescriptors by calling:

```
PropertyDescriptorList = bi.getPropertyDescriptors();
```

This call returns a PropertyDescriptor[] that contains a description of all of the properties contained by this Bean. This includes name, readMethod, writeMethod, type, EditorClass, etc. Our utility uses this information to get the readMethod, writeMethod and property name via the PropertyDescriptor superclass (i.e., FeatureDescriptor) by calling:

```
methodRef =
PropertyDescriptorList[i].getReadMethod();
methodRef =
PropertyDescriptorList[i].getWriteMethod();
propertyName =
PropertyDescriptorList[i].getName();
```

Bean events can be retrieved via EventSetDescriptors by calling:

```
eventSetDescriptorList = bi.getEventSetDescriptors();
```

This call returns an EventSetDescriptor[] that contains a description of all the methods associated with each event for this Bean. Our utility uses this information to get a Method[] for each one of the returned events in the eventSetDescriptorList. This is accomplished by calling:

```
methodList =
eventSetDescriptorList[i].getListenerMethods();
```

This information is used to identify the methods associated to each one of the listener methods.

Information concerning the interfaces implemented by a class can be accessed by calling:

```
interfaceList = c.getInterfaces();
```

This call returns a class[] that contains all the interfaces implemented by the local class. Notice that this call doesn't have a getDeclaredInterfaces() counterpart like the other methods.

The output from the program for examining the events in the class, *com.sun.swing.JPanel* is shown in Figure 5.

```

C:\>java XRayBean com.sun.java.swing.JPanel events

Event Name: ancestor
Listener Method[0]: public abstract void com.sun.java.swing.event.AncestorListe
er.ancestorAdded(com.sun.java.swing.event.AncestorEvent)
Listener Method[1]: public abstract void com.sun.java.swing.event.AncestorListe
er.ancestorRemoved(com.sun.java.swing.event.AncestorEvent)
Listener Method[2]: public abstract void com.sun.java.swing.event.AncestorListe
er.ancestorMoved(com.sun.java.swing.event.AncestorEvent)
-----
Event Name: vetoableChange
Listener Method[0]: public abstract void java.beans.VetoableChangeListener.vetoa
bleChange(java.beans.PropertyChangeEvent) throws java.beans.PropertyVetoExceptio
n
-----
Event Name: mouse
Listener Method[0]: public abstract void java.awt.event.MouseListener.mouseClick
ed(java.awt.event.MouseEvent)
Listener Method[1]: public abstract void java.awt.event.MouseListener.mousePress
ed(java.awt.event.MouseEvent)
Listener Method[2]: public abstract void java.awt.event.MouseListener.mouseRelea
sed(java.awt.event.MouseEvent)
Listener Method[3]: public abstract void java.awt.event.MouseListener.mouseEnter
ed(java.awt.event.MouseEvent)
Listener Method[4]: public abstract void java.awt.event.MouseListener.mouseExite
d(java.awt.event.MouseEvent)
-----
Event Name: key
Listener Method[0]: public abstract void java.awt.event.KeyListener.keyTyped(jav
a.awt.event.KeyEvent)
Listener Method[1]: public abstract void java.awt.event.KeyListener.keyPressed(j
ava.awt.event.KeyEvent)
Listener Method[2]: public abstract void java.awt.event.KeyListener.keyReleas
ed(java.awt.event.KeyEvent)
-----
Event Name: component
Listener Method[0]: public abstract void java.awt.event.ComponentListener.compon
entResized(java.awt.event.ComponentEvent)
Listener Method[1]: public abstract void java.awt.event.ComponentListener.compon
entMoved(java.awt.event.ComponentEvent)
Listener Method[2]: public abstract void java.awt.event.ComponentListener.compon
entShown(java.awt.event.ComponentEvent)
Listener Method[3]: public abstract void java.awt.event.ComponentListener.compon
entHidden(java.awt.event.ComponentEvent)
-----
Event Name: container
Listener Method[0]: public abstract void java.awt.event.ContainerListener.compon
entAdded(java.awt.event.ContainerEvent)
Listener Method[1]: public abstract void java.awt.event.ContainerListener.compon
entRemoved(java.awt.event.ContainerEvent)
-----
Event Name: focus
Listener Method[0]: public abstract void java.awt.event.FocusListener.focusGain
ed(java.awt.event.FocusEvent)
Listener Method[1]: public abstract void java.awt.event.FocusListener.focusLost(
java.awt.event.FocusEvent)
-----
Event Name: propertyChange
Listener Method[0]: public abstract void java.beans.PropertyChangeListener.prope
rtyChange(java.beans.PropertyChangeEvent)
-----
Event Name: mouseMotion
Listener Method[0]: public abstract void java.awt.event.MouseMotionListener.mous
eDragged(java.awt.event.MouseEvent)
Listener Method[1]: public abstract void java.awt.event.MouseMotionListener.mous
eMoved(java.awt.event.MouseEvent)

C:\>

```

Figure 5

## Conclusion

In this article, we took a look at the Reflection and Introspection APIs and used them to develop several useful utilities for Java development. In our next article, we will use the concepts and utilities introduced here to develop a new category of dynamically generated adapters called Dynamic Adapters.

The traditional Adapter design pattern is defined as follows:

**Adapter:** "Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces." [Design Patterns: Elements of Resuable Object-Oriented Software, Gamma et. al., Addison Wesley, 1995.]

Adapters are used when the input and output interfaces are known at compile-time. Dynamic Adapters will allow a pro-

gram to dynamically map the interfaces at runtime. We will examine these patterns in more detail in the next article. ☘

## About the Authors

*Ajit Sagar is a member of the Technical Staff at i2 Technologies, Dallas, TX. He holds an M.S. in Computer Science from Mississippi State University. Ajit focuses on UI, networking and middleware architecture development. He has 7-1/2 years of programming experience, two in Java. Ajit can be reached at Ajit\_Sagar@i2.com*

*Israel Hilerio is a Member of the Technical Staff at i2 Technologies, Dallas, TX. He holds a B.S. in Computer Science Engineering from St. Mary's University, TX and an M.S. in Computer Science from Southern Methodist University, Dallas, TX. He has 8 years of programming experience, 2 1/2 years in Java. Israel can be reached at Israel\_Hilerio@i2.com*



## Listing 1: Cookie.java

```
// Cookie interface and derived classes

interface Cookie {
}

class FortuneCookie implements Cookie{
    public static Cookie newCookie(String str)
    {
        System.out.println("Fortunate " + str);
        return new FortuneCookie();
    }
}

class MisFortuneCookie implements Cookie{
    public static Cookie newCookie(String str)
    {
        System.out.println("UnFortunate " + str);
        return new MisFortuneCookie();
    }
}
```

## Listing 2: CookieFactory.java

```
import java.lang.*;
import java.lang.reflect.*;

public class CookieFactory {

    // Instantiate a Cookie represented by the
    // input String parameter.
    public Cookie createCookie(String className)
    {
        // Get the class corresponding from String
        Class c = null;
        try {
            c = Class.forName(className);
        }
        catch (ClassNotFoundException cnfe) {
            cnfe.printStackTrace();
        }

        // Parameters for the "printCookie()" method
        // of the Cookie interface.
        Class pTypes[] = new Class[1];
        pTypes[0] = String.class;

        // Get "printCookie()" method for Cookie class
        Method method = null;
        try {
            method = c.getMethod("newCookie", pTypes);
        }
        catch (NoSuchMethodException nsme) {
            nsme.printStackTrace();
        }

        // Invoke "printCookie()" method to get Cookie
        Cookie cookie = null;
        try {
            Object[] params = new Object[1];
            params[0] = "Cookie";

            cookie = (Cookie)method.invoke(c, params);
        }
        catch (InvocationTargetException ite) {
            ite.getTargetException().printStackTrace();
        }
        catch (IllegalAccessException iae) {
            iae.printStackTrace();
        }
    }
}
```

```
        return cookie;
    }

    // Simple test for the Cookie Factory
    public static void main(String[] args)
    {
        CookieFactory cf = new CookieFactory();
        Cookie fc = cf.createCookie("FortuneCookie");
        Cookie mfc = cf.createCookie("MisFortuneCookie");
    }
}
```

## Listing 3: XRayBean.java

```
import java.lang.reflect.*;
import java.beans.*;
import java.util.*;

public class XRayBean {

    public Method[]
        getMethodsFromBean(BeanInfo bi) {

        Method[] mList;
        MethodDescriptor[] mDescriptorList;

        mDescriptorList = bi.getMethodDescriptors();
        mList = new Method[mDescriptorList.length];

        for (int i = 0; i < mDescriptorList.length; i++) {
            mList[i] = mDescriptorList[i].getMethod();
        }
        return mList;
    }

    public PropertyDescriptor[]
        getPropsFromBean(BeanInfo bi) {

        PropertyDescriptor[] pDescriptorList;

        pDescriptorList =
            bi.getPropertyDescriptors();
        return pDescriptorList;
    }

    public EventSetDescriptor[]
        getEventsFromBean(BeanInfo bi) {

        EventSetDescriptor[] eDescriptorList;

        eDescriptorList =
            bi.getEventSetDescriptors();
        return eDescriptorList;
    }

    public Method[]
        getListenerMethods(EventSetDescriptor esd) {

        Method mList[];

        mList = esd.getListenerMethods();
        return mList;
    }

    public static void main(String[] args) {
        XRayBean xray = new XRayBean();

        if (args.length != 2) {
            System.out.println("Usage: java XRayClass " +
                "<className> [methods | properties | events]");
            System.exit(1);
        }
    }
}
```



```

}

try {
    Class c      = Class.forName(args[0]);
    BeanInfo bi = Introspector.getBeanInfo(c);

    if (args[1].equals("methods")) {
        Method[] mList = xray.getMethodsFromBean(bi);

        for (int i = 0; i < mList.length; i++) {
            System.out.println("Method[" + i +
                "]: " + mList[i]);
        }
    }
    else if (args[1].equals("properties")) {
        PropertyDescriptor[] pList =
            xray.getPropsFromBean(bi);

        for (int i = 0; i < pList.length; i++) {
            System.out.println(
                "-----");
            System.out.println("Property Name: " +
                pList[i].getName());
            System.out.println("Read Method[" +
                i + "]: " + pList[i].getReadMethod());
            System.out.println("Write Method[" +
                i + "]: " + pList[i].getWriteMethod());
        }
    }
    else if (args[1].equals("events")) {
        EventSetDescriptor[] eList =
            xray.getEventsFromBean(bi);

        for (int i = 0; i < eList.length; i++) {
            System.out.println(
                "-----");
            System.out.println("Event Name: " +
                eList[i].getName());
        }

        Method mList[] =
            xray.getListenerMethods(eList[i]);

        for (int j = 0; j < mList.length; j++) {
            System.out.println("Listener Method[" +
                j + "]: " + mList[j]);
        }
    }
    else {
        System.out.println("ERROR: " +
            "The selected feature is not implemented");
    }
}
catch (java.lang.ClassNotFoundException e) {
    e.printStackTrace();
}
catch (java.beans.IntrospectionException e) {
    e.printStackTrace();
}
}

}

Method[] mList;

mList = c.getDeclaredMethods();
return mList;
}

public Method[]
getAllMethodsFromClass(Class c) {

    Method[] mList;

    mList = c.getMethods();
    return mList;
}

public Constructor[]
getAllConstructorsFromClass(Class c) {

    Constructor[] cList;

    cList = c.getDeclaredConstructors();
    return cList;
}

public Field[] getFieldsFromClass(Class c) {
    Field[] fList;

    fList = c.getDeclaredFields();
    return fList;
}

public Class[] getClassesFromClass(Class c) {
    Vector cList = new Vector();
    Class cTemp;
    Class[] aList;

    cTemp = c;
    while ((cTemp =
        cTemp.getSuperclass()) != null) {
        cList.addElement(cTemp);
    }

    aList = new Class[cList.size()];
    for (int i = 0; i < cList.size(); i++) {
        aList[i] = (Class) cList.elementAt(i);
    }
    return aList;
}

public Class[]
getInterfacesFromClass(Class c) {

    Class[] cList;

    cList = c.getInterfaces();
    return cList;
}

public static void main(String[] args) {

    XRayClass xray = new XRayClass();

    if (args.length != 2) {
        System.out.println("Usage: " +
            "Java XRayClass <className>" +
            "[localMethods | allMethods | " +
            "Constructors | fields | inheritance | " +
            "interfaces] ");
        System.exit(1);
    }
}

```

#### Listing 4: XRayClass.java

```

import java.lang.reflect.*;
import java.util.*;

public class XRayClass {

    public Method[]
        getLocalMethodsFromClass(Class c) {

```



```

}
try {
    Class c = Class.forName(args[0]);

    if (args[1].equals("localMethods")) {
        Method[] mList =
            xray.getLocalMethodsFromClass(c);

        for (int i = 0; i < mList.length; i++) {
            System.out.println("Method[" + i +
                "]: " + mList[i]);
        }
    }
    else if (args[1].equals("allMethods")) {
        Method[] mList =
            xray.getAllMethodsFromClass(c);

        for (int i = 0; i < mList.length; i++) {
            System.out.println("Method[" + i +
                "]: " + mList[i]);
        }
    }
    else if (args[1].equals("Constructors")) {
        Constructor[] cList =
            xray.getAllConstructorsFromClass(c);

        for (int i = 0; i < cList.length; i++) {
            System.out.println("Constructor[" + i + "]: " + cList[i]);
        }
    }
    else if (args[1].equals("fields")) {
        Field[] fList =
            xray.getFieldsFromClass(c);

        for (int i = 0; i < fList.length; i++) {
            System.out.println("Field[" + i +
                "]: " + fList[i]);
        }
    }
    else if (args[1].equals("inheritance")) {
        Class[] cList =
            xray.getClassesFromClass(c);

        for (int i = 0; i < cList.length; i++) {
            System.out.println("Class[" + i +
                "]: " + cList[i]);
        }
    }
    else if (args[1].equals("interfaces")) {
        Class[] cList =
            xray.getInterfacesFromClass(c);

        for (int i = 0; i < cList.length; i++) {
            System.out.println("Interface[" + i +
                "]: " + cList[i]);
        }
    }
    else {
        System.out.println("ERROR:" +
            "The selected feature is not implemented");
    }
}
catch (java.lang.ClassNotFoundException e) {
    e.printStackTrace();
}

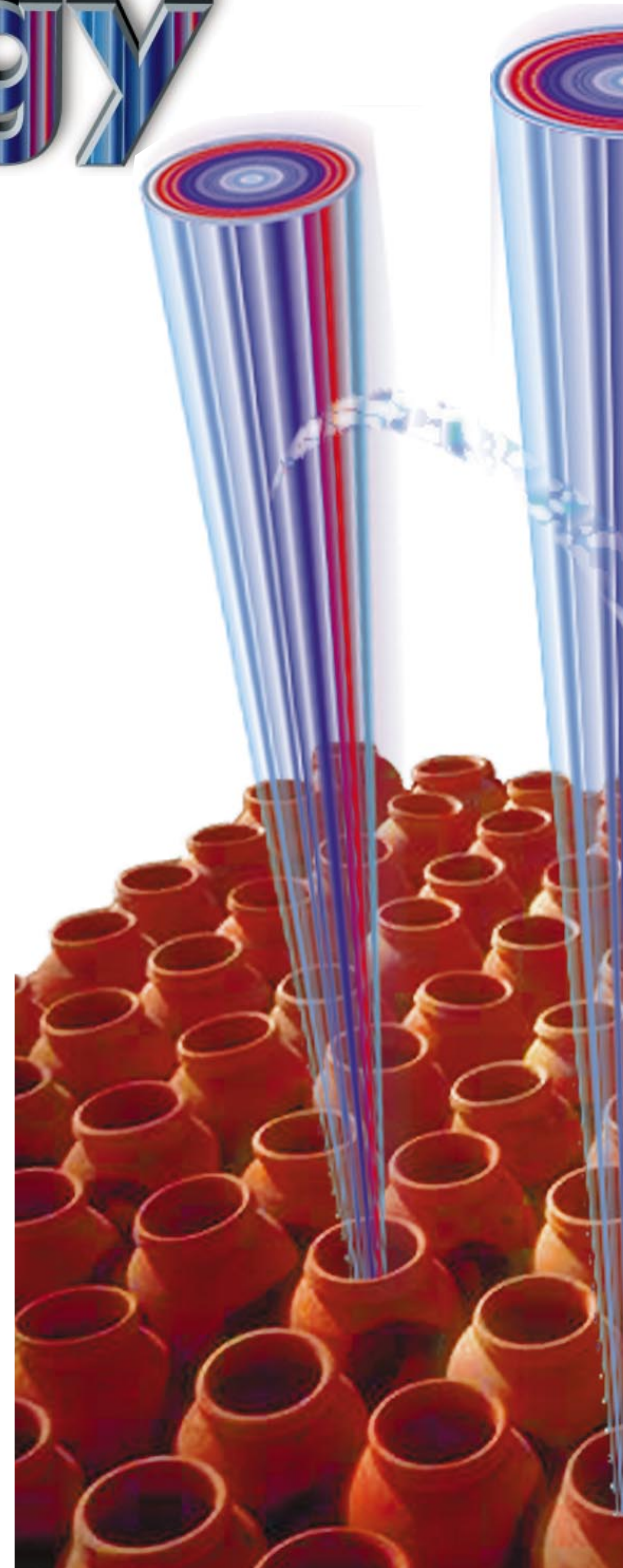
```

# Enterprise Strategy WITH JAVA

*A strong case for using Java  
in component building*

by Graham Harrison

Increasingly, technologists are asked by strategists to state the capability of Java within a distributed component architecture. The larger corporate platform is mixed and the owning, interacting businesses must implement a framework technical architecture in which present and future components can co-exist and change with minimum impact. Larger installations contain data and applications at corporate and departmental levels across a heterogenous computing environment. Technologists, thus, have to articulate some of the values and norms of the business strategist as the business and technology surfaces merge.



This article outlines the typical components in a larger technical architecture framework and explores the contribution Java makes to realizing the key business drivers which underpin it.

## Key business drivers

There are several key business drivers which a framework architecture should address.

### Stable Technical Environment

It is essential that changes in the underlying technical environment do not adversely impact the stability of the core information systems such that achievement of the key business objectives is threatened.

It is unrealistic to expect

technology to remain stable for the foreseeable future. Hence, the architecture must support a method of separating business logic from the underlying technology such that both can be maintained independently of one another.

### Responsiveness to Changing Requirements

Most businesses operate within a very volatile and fiercely competitive environment. It is very important that systems developed within the technical architecture are highly flexible and capable of rapid change. In order to meet this requirement, the architecture should facilitate the ability to easily apply changes to existing systems with the least possible need to develop new, ad hoc applications.

### Cost Effective Maintenance

The level of ongoing operational support and maintenance costs for the new strategic systems must be as low as possible. In order to meet this requirement, the architecture should facilitate maximum reusability of code and enable the business to take advantage of object-oriented techniques and methodologies as they develop and mature.

### Future Proofing

The IT industry is in a constant state of evolution, with innovative technical solutions offering real competitive advantages to businesses that are constantly appearing. It is essential that the technical architecture be flexible enough to be able to integrate these solutions at modest cost as they become available, without compromising existing systems. The way to achieve this objective is to ensure that the architecture adequately defines boundaries between individual technical components and embraces open standards wherever possible such that the components can be replaced relatively easily.

### Supplier Independence

The technical architecture must not be dependent on any one supplier to the extent that there is the potential for that supplier to be able to adversely impact the business in any way. Examples of this would be raising component or service costs to an extreme level, moving away from open standards to more proprietary solutions or by refusing to embrace new technical directions as they emerge.

## The Technical Architecture Framework

The primary purpose of a technical architecture is to ensure that IS systems and services are delivered in a manner consistent with the business requirements. Figure 1 illustrates the technical architecture framework.

The framework comprises a set of discrete components as illustrated in the diagram. Each component has a clearly defined function and they interact via formal interfaces. This approach enables appropriate technical products to be selected for each component and, because the interfaces remain consistent, enables new technology to be easily integrated into the framework as it becomes available.

It is widely recognized within the industry that a three-tier approach should be adopted when building client/server applications, whereby there is formal partitioning between the data, application and presentation layers.

Most interactive Information Systems can be split into two broad categories: On Line Transaction Processing Systems (or OLTP), which provide computerized solutions for business processes such as Payroll, Order Processing, etc.; and, On Line Analytical Processing (or OLAP), which provides management information and decision support facilities. A different set of design considerations and enabling technology is required for each category and needs to be catered for separately within the framework.

The framework also makes a distinction between small departmental or workgroup based applications and large enterprise-wide systems. This is important because there is often a requirement for enterprise data to be made available to departmental systems and appropriate enabling technology must be available to meet this requirement.

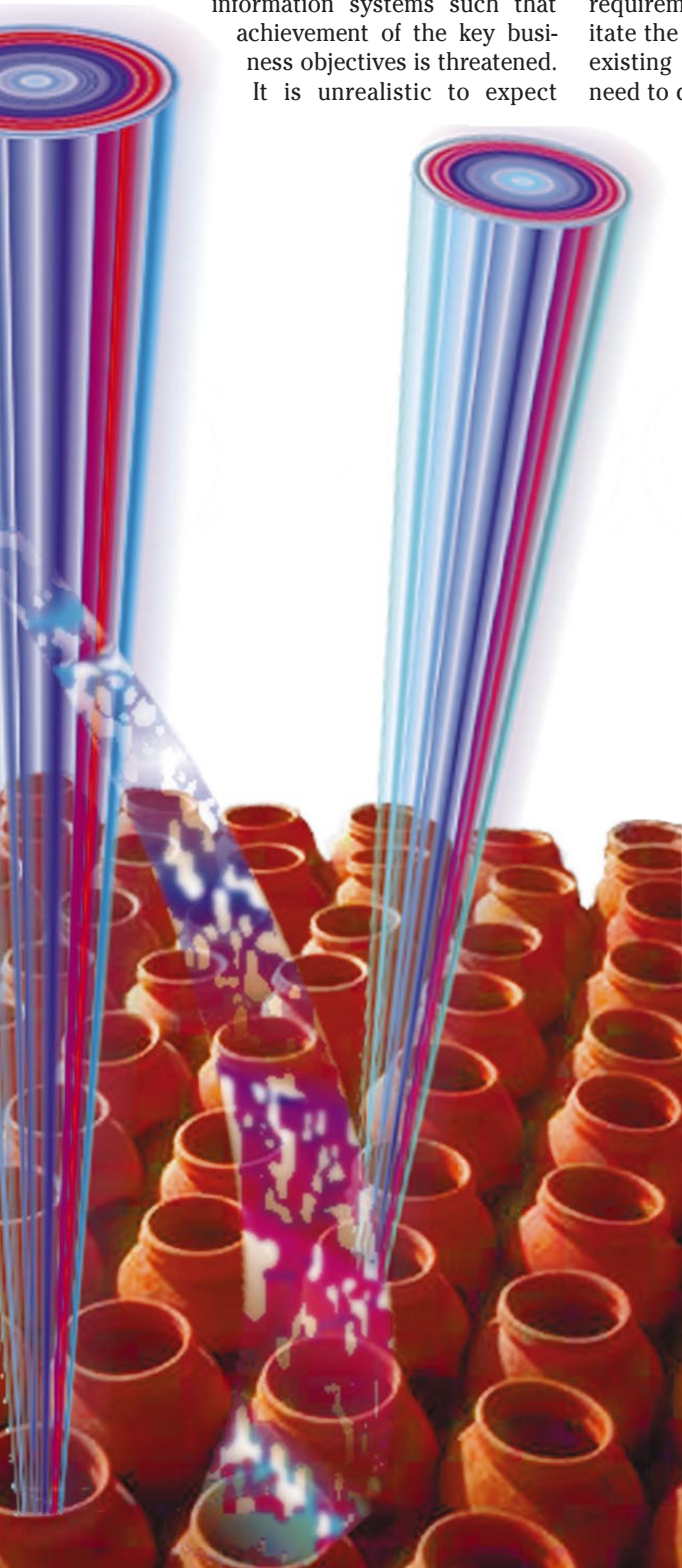
## Architecture Components

This section looks at each component in more detail.

### Corporate Data

This component comprises the corporate databases. A key aspect of the architecture is that all corporate data is stored and managed in a consistent manner. This implies that all information relating to "Customer," for example, is held within the Customer database and that all applications that update Customer details or require Customer-related information would access this single database.

It is important that the business not become too heavily locked into any particular database product. It means that it will be possible to migrate any corporate data



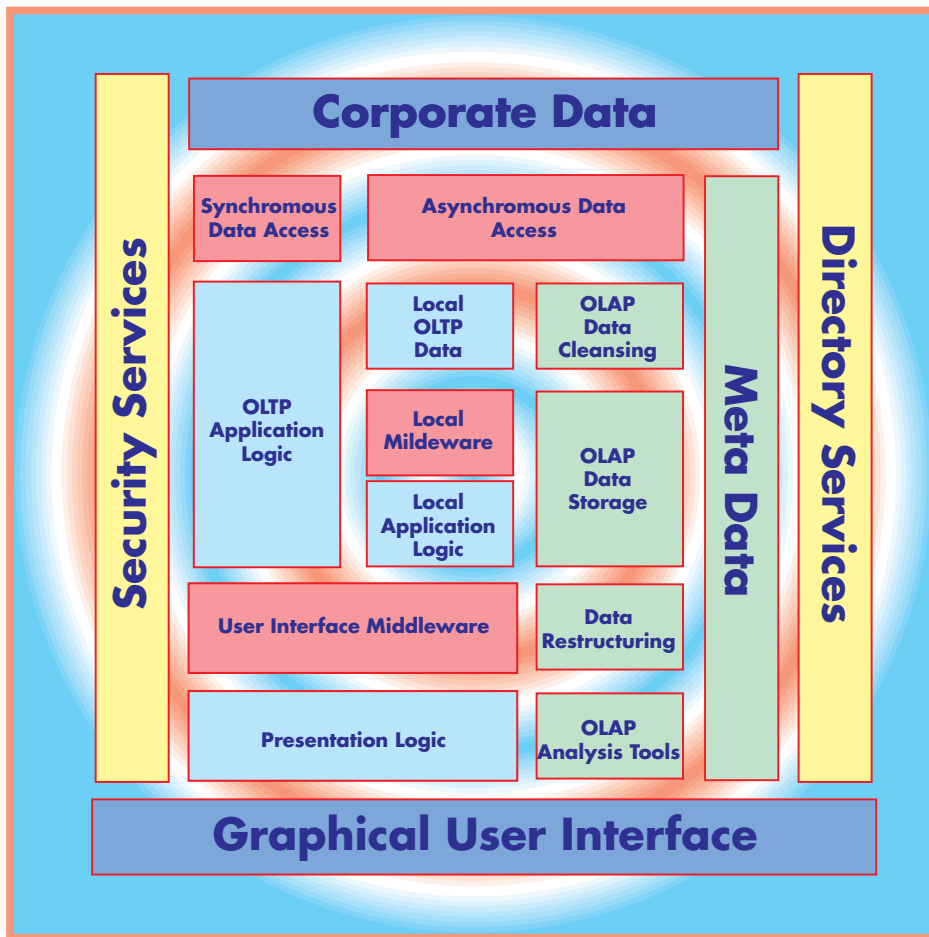


Figure 1: The Technical Architecture Framework

to an alternative RDBMS and/or server platform without changing the application code. Therefore, most installations will insist, for example, on the use of Btree only structures, no user-defined datatypes and no stored database procedures, while maintaining a high level of security, systems administration, reliability, easy access by all strategic applications and access to significant bandwidth and wide area network connectivity.

#### **Synchronous Data Access**

This component relates to the manner in which OLTP applications synchronously access corporate data and represent the top, data access tier in the three-tier model. The main purpose of this component is to provide a formal interface between the business or application logic and the corporate data. This is achieved by placing all data access logic into separate program modules which can then be "called" interactively by the main application programs.

There are various standards for providing synchronous program-to-program communication across a mixed platform; for example, Java RMI and Remote Procedure Call (which is part of the Open Group's Distributed Computing Environment or DCE). Both allow the developer to issue program calls to remote modules as though the mod-

ules were residing on the same platform. Managing the communications protocols and identifying the correct server are all handled by Java or DCE. These facilities, however, are non-transactional in nature. Commitment control is not provided as part of the standard. RPC using Sockets requires the developer to hardcode the location of the target system within the application. Any widespread deployment of Sockets could result in unacceptable management and maintenance overheads.

#### **Asynchronous Data Access**

This component relates to the requirement to asynchronously transfer data between applications or from one database to another. The most common requirement is to transfer data to and from central, corporate databases and multiple, distributed servers. This component is equally applicable to both OLTP and OLAP applications.

Synchronous data access requires both source and target systems to be available for communication in a highly interactive manner. In contrast, asynchronous data access implies that the level of interaction is far less and that the target system does not necessarily even have to be available when the source request is initiated and a requirement to provide queuing facilities may be required. A typical example of asyn-

chronous data access is batch file transfer but it can also be a key enabler of event-driven processing between loosely coupled applications. Adoption of a standard architectural element in this area would be a key requirement. The most appropriate solution to this requirement is Message-Oriented Middleware.

The broad requirement for asynchronous data access can be summarized as the ability to push or pull data between any of the server platforms irrespective of location within the wide area network; the ability to send messages between application programs residing on different systems; provision of time or event-based scheduling on any platform; resilience in terms of automatic error recovery and restart; ability to generate error and diagnostic alerts; provision of an API to enable simple application integration; data replication from one database to one or more copy databases; and, the ability to write bespoke code to extract and insert the data and use a file transfer mechanism such as FTP to transfer the extracted data across the network.

#### **OLTP Application Logic**

This component refers to all physical code within the application with the exception of code related to either data access or screen presentation, and comprises the physical code within enterprise applications which map onto business rules and processes. The key point to note is that traditional application programs tightly couple presentation, business logic and data access logic within the source code. The three-tier approach dictates that there should be formal interfaces between these tiers. This component relates only to the business logic tier. For example, if the application requires an extremely large database, it may be appropriate to place the data on the mainframe but in order to minimize processing costs, implement the application on a HP-UX server.

#### **Local OLTP Data**

This component describes the manner in which local data is stored on distributed, departmental systems. In certain circumstances, there is also a requirement to distribute reference data from central corporate databases into the workgroup environment for local validation purposes.

#### **Local Middleware**

This component describes the manner in which local data will be accessed from local applications. The component encompasses both synchronous and asynchronous access. As with corporate applications, there should be a formal interface between the local database, data access logic and application logic. The Local Middleware component provides data access logic. This component should reside on the



local application server rather than the client.

### **Local Application Logic**

As with OLTP Application Logic, this component refers to all physical code within the application with the exception of code related to either data access or screen presentation.

This component comprises the application logic for local, departmental systems. As with enterprise systems, it relates only to the middle tier of the three-tier model.

### **User Interface Middleware**

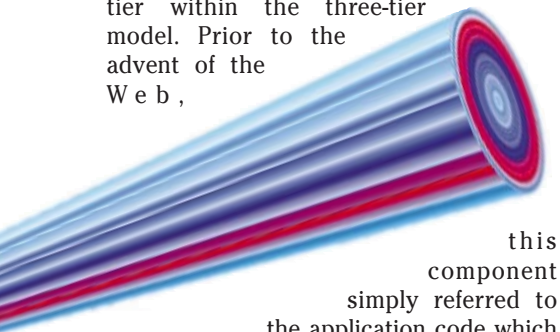
This component refers to the formal interface between the application logic and the presentation layer and is common to both local and enterprise-wide applications.

This component provides the middleware which manages the interaction between both corporate and local application logic as well as the presentation layer within the three-tier model.

Unlike previous layers where there is a possibility that adjacent components could reside on the same physical platform, this component will always involve transfer of information across a wide or local area network between the client and server.

### **Presentation Logic**

This component represents the code that controls how the application communicates with the user via the Graphical User Interface and refers to the technique used to control how the application communicates with the user. This is the bottom tier within the three-tier model. Prior to the advent of the Web ,



this component simply referred to the application code which handled GUI design and screen I/O. Within the context of Internet and intranet applications, this component embraces the use and deployment of Web servers.

### **Meta Data, OLAP Data Cleansing, Storage, Restructuring**

Meta Data is the term used in OLAP to describe all aspects of the data warehouse environment. It covers areas such as: recording of schedules for data acquisition; cleansing and merging operations; recording of process logic for data quality reporting acquisition, cleansing, preparation, distribution, dependencies and hierarchies; recording of users such as data owners,

data editors, end users, security levels; and, recording of infrastructure such as locations of data marts, configurations of user hardware and software. The cleansing component relates to the process of taking data from the post acquisition staging area, validating the relationships between the acquired data and the warehouse as a whole and converting the acquired data into a form suitable for warehousing.

The OLAP storage component defines how cleansed data that is available for OLAP and decision support functions should be stored. The Data Restructuring component relates to the process of preparing stored OLAP data for use via a specific analysis tool. Stored OLAP data is usually held in a de-normalized form. Analysis tools will typically require data to be heavily normalized, summarized and aggregated with a significant use of indexing.

### **OLAP Analysis Tools**

This component defines the specific analysis tools that can be used to manipulate the restructured OLAP data.

### **Graphical User Interface**

This component refers to the common graphical user interface that is to be deployed on all application systems, either traditional windows or Web Browsers, including non graphical interfaces such as hand held devices and automation equipment.

### **Security Services**

The most difficult aspect of providing security services is coming up with a solution that can integrate application security requirements with those relating to securing access to hardware and network resources. Selection of the most appropriate security infrastructure is heavily dependent upon the technical components chosen for the rest of the architecture.

### **Directory Services**

Directory Services refers to the requirement to register all system resources such as clients, servers, file systems, networking details, printers, etc. within a single directory so that they can be accessed from anywhere within the distributed environment. This requirement is essential if the distributed environment is to be managed effectively. The lack of a single solution in this area would mean that the distributed environment is inherently unstable, particularly when any major component change is implemented.

### **Java Support for the Technical Architecture Framework**

Java is a strategic platform in an environment where different applications are supported by different products on different platforms. The reasons can be summarized as follows:

Interfacing code needs to be written once; it will run on all participating hosts that run Java.

Java can encapsulate legacy systems and enable them within the component framework so they can participate in the new multi-tier strategy. These encapsulated legacy systems can then be incorporated into the maintenance infrastructure; e.g., case tools and IDEs using the JavaBean interface. The smallest Java Bean components can be used across all enterprise components, including legacy application components.

The designers can concentrate on the framework components rather than the regression factors in the implementation since VB is not scaleable and C++ has too much build and test overhead. Standard component libraries for C++ tend to be difficult to learn.

Java supports connectivity across all types of framework components without the complexities of C++. In some cases, components can be removed; some JDBC drivers, for example, do not need middleware products such as comms servers to connect to the database over a network.

Java allows thin client application delivery using a browser and is designed for network delivery of code and data to traditional and more exotic devices.

Major vendors are rewriting core products and services in Java or implementing a Java-enabled interface – in particular, Informix (Data Director for Java), Oracle (database, tools) and Sybase (middleware). Sun (and others) is formalizing a portable enterprise component standard called Enterprise JavaBeans™, which will allow an application to make use of whatever enterprise services happen to exist on the platform in use; it makes sense to use the intelligent work done by others.

It is likely that the vendors who supply different parts of the technical architecture will offer products written in Java or supporting Java connectivity. It makes sense to leverage this investment by major vendors and simplify the interface, instead of understanding how to write a C++ module and interface it with a Java enabled database.

Infrastructure services are frequently implemented on different platforms using different products and technologies, making it difficult to build portable application systems. Java provides a common interface to the underlying infrastructure services, regardless of the actual implementation.

For Synchronous Data Access, Java uses JDBC to interface with servers that perform data access. Vendors are shipping JDBC drivers which eliminate the need for network or comms servers between remote clients or application servers and the database

server. This is possible because of the native support in Java for remote procedure call which hides networking functionality. This means there are fewer components to install and maintain. The clear advantage in a Java implementation would be the relative transparency of the remote method call when using RMI. A Java application server can operate with the synchronous data access tier using RMI calls. The data access tier invokes the JDBC API, with or without the relevant bridge. The important point is that what is formally expressed as a rule in the architecture framework is realized but made transparent by the Java platform, which retains the soft link to the implementation detail that can be changed at the lower end without breaking the component interfaces.

It is cheaper to encapsulate legacy applications and databases than to rewrite business logic that doesn't need rewriting. RMI (Remote Method Invocation) runs over the native Java Remote Method Protocol (JRMP) or, in the future, over the industry-standard Internet InterORB Protocol (IIOP). Non-Java clients can invoke an application using CORBA IDL running over IIOP or a COM/CORBA Internet working service running over IIOP. Java can also interface with non-Java languages and applications in the form of standard application calls.

Regarding OLTP Application Logic, the most flexible application delivery vehicle is the browser which delivers the relevant Java classes on demand, with some locality in the Jar archive that can be sent with the Web page. Clearly, the browser would only want to render the presentation layer and the elementary client-validation logic. Code is loaded by the applet on demand from the network resource, effectively the UI and presentation logic; it is scalable due to the ability to parcel up related code in one transmission, including multimedia objects.

With Java, the code is always up to date; when code is updated, only the network repository needs to be updated; no code will persist on the client.

Java-enabled browsers act as front-end viewers. Mediator software accepts requests from the front end, applies whatever business rules are necessary and then relays the request to the enterprise data systems at the back end. Under this model, application distribution is automatic and requires no additional software beyond a Java-enabled Web browser.

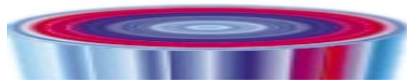
The options for connecting the thin, browser client to the application server are TCP/IP sockets or remote procedure call using RMI.

The ability to Web-enable legacy applications and encapsulate them as a Java-

Bean allows them to be portable to a number of different server platforms without reprogramming. Beans can communicate with other object models, such as CORBA, OpenDoc, ActiveX or OLE. Individual components can be inserted, updated or deleted with minimal disruption to the application architecture. This multi-tiered approach allows clients to communicate with older or proprietary server applications encapsulated within JavaBeans and allows legacy systems to participate in a more dynamic and integrated development platform. Instead of coding an interface to



*For local middleware, it is easier to manage RPC with RMI than in other languages simply because the Java runtime manages the interfacing.*



manage differences, the development IDE which implements Beans can use the encapsulated system.

Applications are thus inherently scalable and can run in a multithreaded, multi-processing environment, and Beans may be reusable for new applications, reducing development time.

For local middleware, it is easier to manage RPC with RMI than in other languages simply because the Java runtime manages the interfacing. Application servers which have to manage an interface against different versions of Java will use sockets. Again, the complexity of socket connections is supported by a standard API in the language and is easier to use than C/C++. IDEs (Integrated Development Environments) for Java currently support an 'n-tier' model at design time and some are shipping with application servers, so the elapsed time for managing these interfaces is reduced because the IDE is managing the interface component and integration earlier in the build cycle. This helps both functional (look-and-feel, business function) prototyping and non-functional (back-end) prototyping.

With the PersonalJava API, applications can support a variety of client devices, including telephones, kiosks, smartcards or other Internet-enabled appliances. The exotic nature of these devices is ideal for Java, since it will allow them to participate in the broader framework, interfacing uni-

formly with local or remote architecture components (legacy and otherwise). In this sense, a technical architecture framework is an abstraction in which its core properties define the secure corporate boundary and express the ability to interact with other participating components in an extranet context.

Note that some JDBC implementations already subvert the need for middleware in those database products which currently have it. For example, servers such as SQL/Net (Oracle) or Ingres/Net will not be required once the vendors release 'native' JDBC drivers which connect directly to the server.

Regarding Security and Directory Services, the Java language has a built-in security model; it does not support pointers and has bounds checking. Java also has Byte Code Verification – downloaded classes are checksummed, optionally encrypted and signed. No illegal opcodes are allowed and stack integrity (under or overflow) is also managed by the Java VM. The VM also has the Sandbox. The security manager prohibits network classes from overloading Java base classes and prevents access to files/sockets. The user can also define authentication and encryption of Java applets. Java has a standard library called JNDI (Java Naming and Directory Interface) which can be used to declare and monitor components in the architecture.

## Summary

The explanation above, together with the specifics in the Java platform (language design plus core classes), make a strong case for using Java in component building. This is due to the fact that it is much easier to build and debug than C++ and VB, and also due to how much leverage it has against existing applications, application delivery and integration.

The business metrics of portability, flexibility and separation of business function from contingent technology are clearly met by a Java implementation of a strategic architecture framework. Enterprise JavaBeans is the next exploration of this surface, which includes a transaction-processing element. In a landscape where two-tier client/server applications will be declared legacy applications by 2001, Java is the key enabling technology that will allow the strategist's price/performance ratio to be met. ☉

### About the Author

Graham Harrison is a Senior Consultant with Informix and a Sun Certified Java Programmer. He can be contacted at [gpharrison@compuserve.com](mailto:gpharrison@compuserve.com)



[gpharrison@compuserve.com](mailto:gpharrison@compuserve.com)

# The Past, Present and Future of Java Foundation Classes

## *Raising the bar for GUI functionality*

In 1995, Java technology shook the World Wide Web as a network-centric, object-oriented language providing client-side processing that helped Web developers turn otherwise static pages into dynamic visual experiences. Key to the creation of these animated Web pages was Java's graphical user interface library, the Abstract Window Toolkit (AWT).

Now, Sun's Java Developer's Kit (JDK) release 1.1 introduces the first installation of the most significant cross-platform graphical user interface technology since the advent of windowing systems: the Java Foundation Classes (JFC). JFC is a complete graphical user interface (GUI) toolkit that dramatically extends the original AWT with a comprehensive set of classes and services.

JFC is a scalable, robust and open technology that enables developers to create and deploy commercial-grade intranet and Internet applications, casting Java GUI development in a new light. Even as components and services grow, JFC promotes ease of use and facilitates rapid application development. It is delivered as core Java technology, which means it is available on all Java platforms, resulting in faster application downloads, more reliable applications and simplified application deployment.

### History of Java Foundation Classes

When it was introduced, the AWT provided developers with a rudimentary library for building applications and applets – Java applications that are executed from inside a browser instead of being launched and run from the native operating system.

Designed for simple, Web-centric tasks, developers encountered limitations with the AWT when attempting to create modern, sophisticated client applications. Although the AWT was limited in scope, it did offer two important features for all applets and applications:

- 100 percent portability from a single set of source code
- Assumption of a native look and feel on deployment platforms

The AWT delivered on the promise of a standards-based platform that adapted to the user desktop. It also provided a good starting point for graphical Java development with room for improvements – some evolutionary, and some revolutionary.

GUI developers count on baseline functionality to create professional-quality applications. The AWT, while best suited for applet development, provided little integration into the desktop environment and even less functionality for creating large-scale applications. JFC, which delivers a more robust framework for GUI development, also delivers the baseline components and frameworks that developers have come to expect from the Java platform.

### Current State of JFC

JFC extends the original AWT by adding a comprehensive set of GUI class libraries that is completely portable and delivered as part of the Java platform. In addition, JFC will include many of the key features found in Netscape's Internet Foundation Classes. Since the JFC is core to the Java platform, it eliminates the need to download special classes at runtime and is compatible with all AWT-based applications.

JFC includes a rich suite of high-level components and services that are fully and uniquely cross-platform compatible, and offers significant performance improvements. With JFC, developers can create and deploy large scale, mission-critical intranet, Internet and Crossware applications. And because Java is an open, standard technology, a broad complement of third party tools and components are available to enhance application development.

In short, JFC includes many new, easy-to-use and sophisticated features that are

designed to work together to offer the following key advantages over other frameworks:

- JFC is core to the Java platform and reduces the need for bundled classes.
- All new JFC components are JavaBeans.
- JFC has no framework lock-in, so developers can easily bring in other third-party components to enhance their JFC applications.
- JFC components are cross-platform.
- JFC enhanced services promote development of feature-rich applications.
- JFC subclasses are fully customizable and fully extendable.

Additionally, JFC offers:

- JavaBeans compliance
- Lightweight UI framework
- Delegation event model
- Printing
- Data transfer/clipboard
- Desktop colors integration
- Graphics & image enhancements
- Mouseless operation
- Popup menu
- ScrollPane container

### Future of JFC

JFC will continue to expand, with plans in the works to include a rich complement of high-level components that will enhance the user's visual experience and improve user productivity. New application services are slated for JFC that will further integrate Java applications into the desktop environment.

New features will further enhance a developer's ability to deliver scalable, commercial-grade applications. These features will be made available to developers as they are completed and then rolled into the next release of JDK. They will include:

- Drag and drop
- New high-level components
- Pluggable look and feel
- 2-dimensional API
- Accessibility features for the physically challenged

The JFC raises the bar for GUI functionality in Java while delivering a rich API and a growing set of components and services to make it easier for developers to create and deploy commercial-grade applications. ☉



# JavaMedia Newsletter

# JavaMedia Newsletter

# JavaMedia Newsletter

# JavaMedia Newsletter



# TopLink

## by The Object People

*Allowing the end user to store Java objects  
in relational database tables*

by Ed Zebrowski



The distributor had just bought out a few rival businesses in town. The paperwork had been signed, the funds were transferred and the deal was complete. The only detail to be worked out was to link all the locations together into one coordinated unit. The MIS manager was horrified to learn that the databases of the other locations are as diverse as can be. Some of them are relational databases, whereas the home database is of the newer object type. What he needs is a tool that allows him to “wrap” all the databases together into one manageable, tightly knit application. What he needs is TopLink from The Object People.

TopLink is a powerful new application that allows the developer to work at the object level, even if the database in question is relational. The objects are mapped to the database in a non-intrusive manner; therefore, the relational database can be made to behave much like an object database.

### Why Not Just Change the Database from Relational to Object?

There are a few good reasons why companies are not willing to do this at this time. One is that relational database technology is regarded as the standard. Accessible through a wide range of applications, the relational database is the one most working professionals are familiar with. The object database, although it has come a long way in the past few years, is still regarded as new, unfamiliar and untested technology. Even if an organization wanted to make the change, the time and expense of re-encoding data and training personnel would scare them away from this option.

### TopLink Provides the Perfect Solution for this Dilemma

TopLink allows relational databases and objects to be “bridged” together in a cost-effective manner. If object database technology someday becomes the new standard (some think it will), TopLink can allow an easier migration to this change, saving a lot of time and expense in the process.

### What is TopLink?

Basically, TopLink provides a flexible mechanism that allows the end user to store Java objects in relational database tables. It is a layer of Java code between a Java application and the relational database being used. It acts as an object-orientated “wrapper” around the relational database and allows “mapping” of objects into a series of relational tables. This wrapper protects the application from changes in the underlying database. Changes in the database should have no effect on the code in an application.

### Installing TopLink

TopLink installation requires:

- A JDBC driver which can connect with the local database system
- A version of Java which is compatible with JDBC API

If your system has an up-and-running Sun JDK 1.1 or higher, it should meet these requirements.

Installation from the CD-ROM was basic and took only a few minutes. After installation is complete, it is necessary to perform a few steps:

- The local JDBC drivers must be tested – This is done by entering database login information into some code that is supplied with the installation. The code must then be compiled and executed. If all goes well, a message reading: “Successful Disconnection. Test Complete” will be displayed.

### TopLink

The Object People

885 Meadowlands Drive Suite 509

Ottawa, Ontario, Canada K2C 3N2

Phone: 613 225-8812

Fax: 613 225-5943

Web: <http://www.objectpeople.com>

Email: [info@objectpeople.com](mailto:info@objectpeople.com)

Price: \$4,000 per developer

played.

- Configure the local Java environment – Configure development environment parameters which are IDE specific.
- Set the appropriate CLASSPATH variable – I ran the application in WIN 95 using JDK 1.1.5, so my CLASSPATH was set to the following:  
TopLink\Classes  
TopLink\Classes\TopLink.zip  
TopLink\Classes\Tools.zip
- Run a final installation test – This is done by compiling and running some more of the supplied code. A message that reads: “Connection successful” will indicate that all is well.

### Using TopLink

I found installation and configuration of TopLink to be smooth and uncomplicated. Now that the application is up and running, let's look at what it can do. I'll use the example supplied with the application to demonstrate some of the steps necessary to create a simple database application.

The sample included with the software is an employee management database. It keeps track of information for all full-time and contract employees, including addresses and phone numbers. The object model being built consists of three classes:

1. The Employee class represents all employees, both full time and contracted. It includes all personal information, including references to home addresses and phone numbers.
2. The Address class represents home address. This includes country, street,





city and zip code.

3. The Phone number class contains, as you may have guessed, the phone number including area code.

Each of the above classes contains cross-references to the others. The relational database stores all of this information in three tables, much in the same fashion that I've built the object model. It is important to point out here that it's not necessary to structure the object model after the table structure of the relational database. The object model can be based on application requirements rather than the database structure. This is a great feature, as the developer is not bound to the relational database structure.

TopLink communicates with a relational database via a Database Session. This Java class keeps track of the following information:

- Descriptors – Maintain the associations between tables and Java classes
- Identity Maps – Used to cache and maintain identity
- Database Accessor – Handles low level communication between the session and the relational database

## Logging into the Database

In order to login to a database, it is first necessary to create a JDBCLogin instance. This holds the login parameters including the database name and type. This class is then passed to the DatabaseSession, which allows the login to occur. Here's a small fragment of the code that allows the creation of a login instance:

```
import TOPLink.Public.PublicInterface*;
import TOPLink.Public.Exceptions*;
Project project = TOPLink.Public.
    PublicInterface.Project.read(
ic:\\TOPLink\\BuilderDemos\\Employee.
projecti)
```

Now that the database is logged into, it's time to discuss the initialization of the descriptors. This is done by the following steps:

1. Create a Project using the TopLink Builder tool. This is a stand-alone application used to create and manage the descriptors and mappings. It's GUI interfaced and looks like your typical drop-menu driven application. It obtains data from the database and stores it in Table files. Class Definition files generated from the persistent Java classes are supplied, and the Builder is then used to map the table columns to class attributes. The Builder then writes Descriptor files to these mappings. These are used by the Java application to access the database.

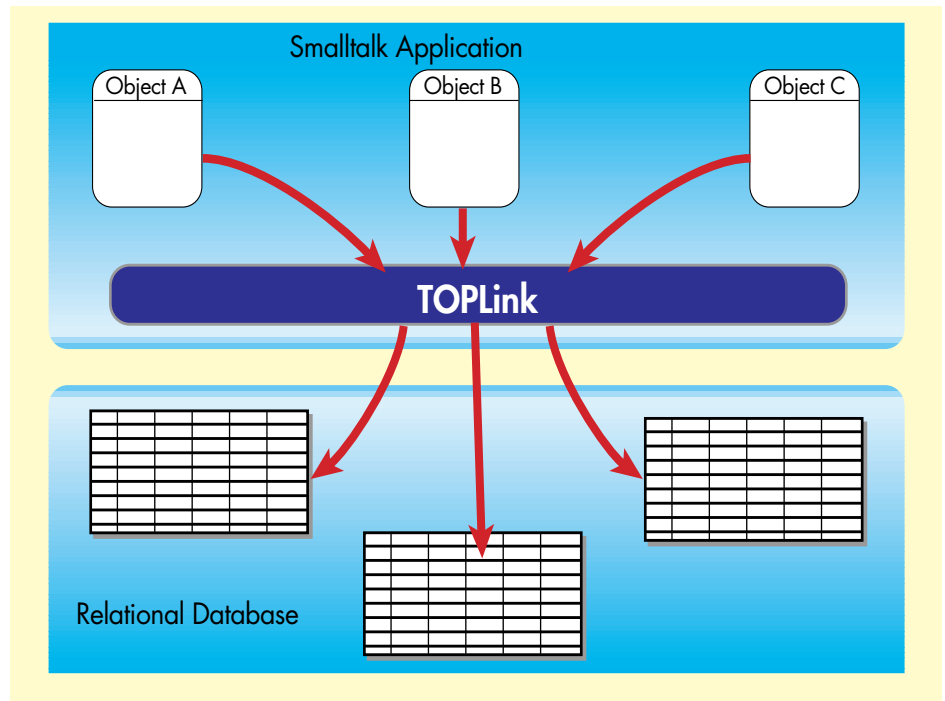


Figure 1: TopLink takes the relational database, and allows it to be treated as and object database

2. If required, modify your persistent Java classes. It is required that all persistent classes have a public default constructor which has no arguments. If any of the class attributes are declared private, it is necessary to create accessor methods to allow them to be written to the database.
3. Create Class Definition Files for each of the persistent Java classes. Each of the Java classes in the project must have a descriptor. Opening the Builder and selecting "Add/Update" classes from the Descriptor menu can create these.
4. Use the Builder's Database Login screen to connect to the database. This is done by selecting "Login To The Database" from the Builder menu. A dialogue box will drop open, allowing database connection information to be entered.
5. Import table information from the database to the project. This is also done through the builder. All that's involved here is selecting the "Add Existing" from the Tables menu.

All in all, it really hasn't been too difficult up to this point. There has been a little filling in the blanks of code, and then compiling to run, but most of the major jobs were done by clicking on menus. That's pretty amazing, when you consider just how potent an application this is. This fraction of an example only begins to scrape the surface of just how many uses there are for this powerful builder.

It is also possible to develop three-tier applications using TopLink. It supports the following databases (and many others through JDBC drivers):

- Oracle
- Sybase
- DB2
- Microsoft SQL Server

Mainframe connectivity is also available for legacy data.

TopLink is one of the more powerful database tools I've seen. It offers flexibility and relative user-friendliness for this type of application. For those of you looking for a powerful tool to develop Java applications by using relational databases, or for someone who wants a good tool to develop a three-tier environment, it is definitely a must see.

I remember when I first discovered Java. At first I thought it was this neat little scripting language that jazzed up Web pages. I almost immediately discovered that I had underestimated its power. I watched that language, in a relatively short period of time, become a major competitor of C, the age-old deity of programming languages. Now, it seems Java is invading the database management and development platforms as well. Will Java take over these traditional applications, and become the dominant development environment of the next century? That remains to be seen, but applications like TopLink provide a good clue as to what just might happen.

## About the Author

Edward Zebrowski is a technical writer based in the Orlando, FL area. Ed runs his own Web development company, ZebraWeb, and can be reached on the Net at zebra@rock-n-roll.com



zebra@rock-n-roll.com





# ADVERTISER INDEX

Advertiser		Page	Advertiser		Page	Advertiser		Page			
<b>Borland</b>	www.borland.com	408 431-1000	59	<b>MindQ</b>	www.mindq.com	800 646-3008	15	<b>Slangsoft</b>	www.slangsoft.com	972 3-7518127	31
<b>Bristol Technology</b>	www.bristol.com	203 438-6969	75	<b>ObjectShare</b>	www.objectshare.com	800 973-4777	21	<b>Socket Software</b>	www.socket.com	814 696-3715	33
<b>Coriolis</b>	www.coriolis.com	800 410-0192	81	<b>Object Matter</b>	www.objectmatter.com	305 718-9101	50	<b>Stingray Software Inc.</b>	www.stingsoft.com	800 924-4223	2
<b>Data Representations</b>	www.datarep.com	203 3633-0160	19	<b>ObjectSpace</b>	www.objectspace.com	972 726-4100	4	<b>SunTest</b>	www.suntest.com	415 336-2005	4
<b>Halcyon</b>	www.halcyonsoft.com	888 333-8820	37	<b>PreEmptive Solutions</b>	www.preemptive.com	216 732-5895	57	<b>Sybex Books</b>	www.sybex.com	510 523-8233	63
<b>ILOG</b>	www.ilog.com	415 688-0200	69	<b>Progress/Cohn &amp; Godly</b>	www.apptivity.com	800 477-6473	23	<b>The Object People</b>	www.objectpeople.com	919 852-2200	47
<b>IBM</b>	www.ibm.com/java	800 IBM-7080	6 & 7	<b>Progress Software</b>	www.protospeed.progress	800 477-6473	27	<b>Thought, Inc.</b>	www.thought.com	415 836-9199	45
<b>Inno Val</b>	www.innoval.com	914 835-3838	49	<b>ProtoView</b>	www.protoview.com/java	800 231-8588	3	<b>Visionary Solutions, Inc.</b>	www.visolu.com	215 342-7185	50
<b>Keo Group</b>	www.keo.com	978 463-5900	61	<b>Roguewave</b>	www.roguewave.com	800-487-3217	17	<b>WebMethods</b>	www.wbmethods.com	888 831-0808	55
<b>KL Group Inc.</b>	www.klg.com	800 663-4723	11	<b>Sales Vision</b>	www.salesvision.com	704 567-9111	13	<b>Zero G. Software</b>	www.zerog.com	415 512-7771	53

1/4 Ad

1/4 Ad



# Instant Basic for Java

## by Halcyon Software, Inc.

*Its easy to add multimedia to your Website*

by Ed Zebrowski



A few years back, I dropped in on a friend who was busy at work on her computer. "Whatcha doin'?", I asked playfully.

"I'm writing a program", she replied without looking up.

As I looked over her shoulder I was befuddled by what I saw. She was opening drop menus and clicking on options or typing simple addresses into dialogue boxes. I was familiar with the monstrous task of typing complicated code in a text editor, and then going through the painful ritual of compiling and debugging.

She said this was a new application called "Visual Basic". It allowed the developer the luxury of staying at a GUI interface, and required no complicated debugging or linking.

Now, it seems, there's this new kid on the block – I think his name is "Java" – and there are a growing number of clients who are demanding applications that are built on his platform. To tap into this lucrative new market, it becomes necessary to learn a whole new programming language. This, as most of you know, is a very tedious and time-consuming adventure. Tedious? Time-consuming? Not anymore! Now there is an amazing new building application from Halcyon Software, Inc. called Instant Basic for Java. Not only does Instant Basic for Java (IB4J) allow development of Java applications in a VB-like architecture, it also enables the developer to migrate existing VB applications to Java platforms.

### Installation

IB4J installs easily on any 90MHz or better Pentium (or SUN SPARC and other Java platforms) with 24 MB of RAM (32MB recommended). It's Pure Java, and therefore will need a functional JDK 1.1.5 or higher. I installed my demo by setting a classpath to the stored directory, and the rest went down as smooth as gravy. It ran on my Cirrus 150 mh2 system with no trouble at all.

### Oh My! It Reminds me of VB!

As I began to use this product, a lot of stuff I thought was lost to memory started to come back. I dare say that if I had been using VB all along, I wouldn't have to consult a great deal of documentation to get started. IB4J features full support for most BASIC Functions, Statements and Objects. Full compatibility with the BASIC language syntax is also provided.

### Java All the Way

Don't be misled by what you've read so far. Although IB4J looks and acts (somewhat) like VB, it is not just some VB knockoff. This is a powerful, stand-alone Java-based builder through and through. A quick examination of a few of its features confirms this. IB4J comes with a Form Painter, a Source Code Editor (for those of you who must type code!), a Project Browser, a Menu Editor, a Compiler and a Graphical Debugger. This tool can help you make the simplest of applets or the most complex Java applications with the bare minimum of time and effort.

### What About my Existing VB Work?

The version of IB4J I worked with (the professional edition) is bundled with a feature called the "Instant Converter". This is a clever device that enabled me to migrate my existing VB applications to the Java language. It supports all Microsoft VB Datatypes including variant and object. Existing DAO code is transparently migrated into Java using JDBC. I experienced no real difficulty in converting any of my forms, models or classes into Pure Java classes.

### What About Database Development?

Some of you may develop from and maintain a large database. IB4J can be a powerful and useful tool for you also. Bundled with the professional edition, is another feature, Cloudscape's evaluation version of JBMS, a leading Java-SQL Object-Relational Database system for high-performance database development.

### Instant Basic for Java

Halcyon Software, Inc.

50 W. San Fernando St. Suite 1015

San Jose, CA 95113

Phone: 408-998-1998 Fax: 408-998-1922

Web: <http://www.halcyonsoft.com>

Email: [info@halcyonsoft.com](mailto:info@halcyonsoft.com)

Price: \$99 standard edition,  
\$795 professional edition (tested)

Support for Data Controls, Data Access Object and Remote Data Access Object is included. This support is built using a standard JDBC interface. This couples with many popular SQL's like Oracle, Informix and Microsoft SQL.

### It Can Even Help with Your Install Challenges

The professional edition I worked with came equipped with another neat feature: Instant Installer. This permits you to create single-step installation files that can be executed on any Java-enabled platform. It provides support for various installation features, like serial number generation, creating and saving of multiple distribution configurations, and even those "Compact, Typical or Custom" installation options we've all come to know and love. This can generate a really professional installation wizard that can display promotional side-screens and even go as far as to provide JVM auto-detection as well.

IB4J is really a slick tool. It has invited another group of people, VB developers, to jump right into the warm, inviting waters of Java development. If you are VB proficient (and even if you're not) and you want to see what this Java thing is all about, put Instant Basic for Java on your "must have" list.

### About the Author

Edward Zebrowski is a technical writer based in the Orlando, FL area. Ed runs his own Web development company, ZebraWeb, and can be reached on the net at [zebra@rock-n-roll.com](mailto:zebra@rock-n-roll.com)



[zebra@rock-n-roll.com](mailto:zebra@rock-n-roll.com)



# Developing Distributed Applications in CORBA: A Tutorial

## *Anatomy of a CORBA-based application*

by Qusay Mahmoud

### Introduction to CORBA

CORBA, which stands for Common Object Request Broker Architecture, is an industry-standard developed by the Object Management Group (OMG), a consortium of more than 500 companies. CORBA is actually a specification for creating and using distributed objects. CORBA objects are different from typical programming language objects in three ways: CORBA objects can run on any platform; they can be located anywhere on the network; and they can be written in any language that has IDL mappings.

Figure 1 shows a request being sent by a client to an object implementation. The client is the entity that wishes to perform an operation on the object, and the object implementation is the actual code and data that implements the object. The Object Request Broker (ORB), which is the heart of CORBA, is responsible for all the mechanisms required to: (1) find the object implementation for the request; (2) prepare the object implementation to receive the request; and (3) communicate the data making up the request.

The CORBA specification is nothing if there is no software that implements it. The software that implements the CORBA specification is called an Object Request Broker (ORB). There are a number of CORBA implementations that exist in the market today. The most popular ones are: ORBIX from IONA Technologies, VisiBroker from Visigenic Software and JavalDL from JavaSoft. Note that the latest version of Netscape Communicator comes with the VisiBroker ORB embedded into it.

As with some other distributed programming environments (e.g., RMI), CORBA objects are specified with interfaces – the contract between the client and the server. In the case of CORBA, however, the interface is specified in a special language known as the Interface Definition Language or IDL.

### What is IDL?

IDL defines the types of objects by defining their interfaces. An interface consists of a set of named operations and the parameters to those operations. It is important to note that IDL is used to describe interfaces only, and not implementations. That is to say, IDL is not a programming language; however, its syntax is similar to C++ and Java.

Through IDL, a particular object implementation tells its potential clients what operations are available and how they should be invoked.

From the IDL definitions, the CORBA objects are mapped into different languages. Some of the languages that have IDL mappings include: Java, C, C++, Smalltalk, Lisp and Python. In this article we are concerned only with the Java mapping. The mapping includes definitions of Java-specific types and method interfaces to access objects through the ORB. It also defines the interaction between object invocations and the threads of control in the client or implementation.

The scope of this article doesn't allow me to discuss IDL and its mappings to Java; that subject deserves a book on its own. However, since most readers have some knowledge of Java, I think the best way to teach you a little IDL is through analogy. Table 1 shows IDL constructs and the equivalent Java constructs.

In Table 1, note that the IDL module construct is mapped to a Java package. All IDL types within the module are mapped to Java classes or interfaces. So, for example, the following IDL definition:

```
// Sample.idl
module Sample {
    const long dist = 23456;
}
```

is mapped to:

```
// Sample/dist.java
```

This column, as you know, has historically focused on the combination of Java – an immense step forward for portability – and CORBA – the de facto standard for integration of enterprise-wide distributed applications. In this issue we step back and give an overview of CORBA, its genesis and structure, with examples.

Most of the articles I have seen in CORBACorner are aimed at developers who are already familiar with CORBA. This month, I offer a brief practical tutorial that gives an overview of CORBA and then concentrates on the anatomy of a CORBA-based application through the development of a sample application.

### Richard Soley

Editor, CORBACORNER

President and Technical Director of the Object Management Group, Inc.

```
package Sample;
public final class dist {
    public final static int value =
        (int) 23456;
}
```

The application developed here will give you a better understanding of the IDL mapping to Java. This application was developed using VisiBroker 3.1 for Java, which is an ORB with a complete implementation of the CORBA specification.

### Anatomy of a CORBA-based Application

We will use the following steps to develop our application:

1. Describe interface(s) using IDL
2. Implement the CORBA classes
3. Develop the Server
4. Develop the Client
5. Start the smart agent, the server and the client(s)

We will talk about each one separately by walking you through the development of an Arithmetic Server.



## Arithmetic Server

The Arithmetic Server we will develop in this article will be capable of performing arithmetic operations (addition, subtraction, multiplication, etc.) on arrays of integers. However, for demonstration purposes, we will show only the addition operation. As an exercise, modify the code by adding more operations.

The first step in developing the Arithmetic Server is to define one or more interfaces in IDL.

### 1. Defining an Interface

When defining an interface, think of what kind of operations are needed and what parameters the operations should have. Listing 1 shows an IDL interface for the Add object. Note that the `sum_arrays` operation here has three parameters. The first two are the input arrays; the third will hold the result of the sum. Also, note that the first two parameters are declared as “in” and the third is declared as “out”. IDL defines three parameter passing modes: in, out and inout. As the names imply, the in parameters are used for input, the out parameter for output and the inout parameter for both input and output.

Once we finish defining the IDL interface, we are ready to compile it. VisiBroker for Java comes with the IDL compiler `idl2java` which is used to map IDL definitions into Java. We run `idl2java` from the command line and we feed it the IDL interface or module we want to compile, as shown in Listing 2.

As you can see, it has generated a number of files. The important ones to use in this article are:

- `Add.java`: The Arith interface declaration
- `AddOperations.java`: Declares the `sum_arrays` method
- `_st_Add.java`: Stub code for the Arith object on the client side
- `_sk_Add.java`: Stub code for the Arith object implementation on the server side
- `_example_Add.java`: Code you can fill in

IDL	Java
<code>/* comment */</code>	<code>/* comment */</code>
<code>// comment</code>	<code>// comment</code>
<code>module</code>	<code>package</code>
<code>interface</code>	<code>interface</code>
<code>const</code>	<code>public static final</code>
<code>boolean</code>	<code>boolean</code>
<code>char</code>	<code>char</code>
<code>string</code>	<code>String</code>
<code>long</code>	<code>int</code>
<code>float</code>	<code>float</code>
<code>double</code>	<code>double</code>

Table 1: IDL mapping to Java

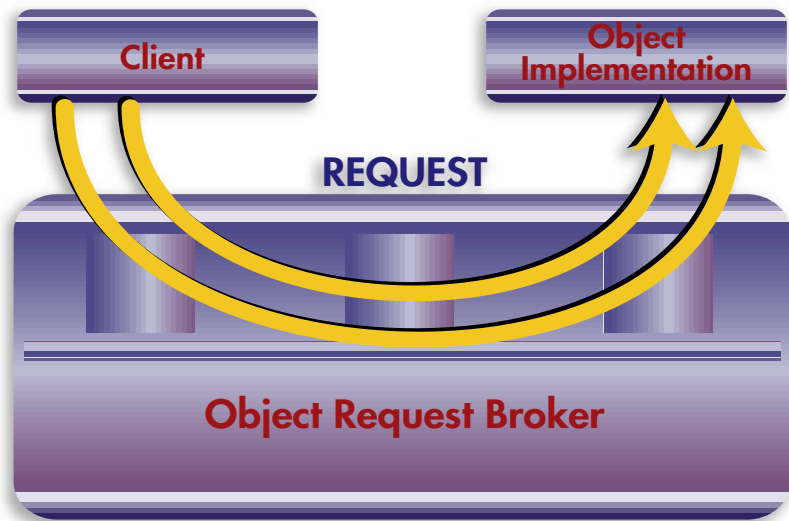


Figure 1: Request sent by client to object implementation

to implement the Arith object on the server side

The `Add.java` file contains the Java code generated from the `Arith.idl` definition. We show the generated code from the IDL definition, stripped of its comments, in Listing 3. This code will help us implement the `sum_arrays` operation.

### 2. Implementing the CORBA classes

Implementing the `sum_arrays` operation is really easy. As shown in Listing 2, the `idl2java` compiler has generated a file named: `_example_Add.java`. This file actually contains some constructors as well as the definition of the `sum_arrays` method. I copied the file into a new file: `AddImpl.java` and implemented the `sum_arrays` method as shown in Listing 4.

Now we can compile it:

```
% javac AddImpl.java
```

We are ready now to develop our server program.

### 3. Develop the Server Program

Listing 5 shows the implementation of the Server class for the server-side of the Arithmetic Server application. The Server class does the following:

- Initializes the Object Request Broker
- Initializes the Basic Object Adapter (BOA)
- Creates an `AddImpl` object
- Activates the newly created object
- Prints out a status message
- Waits for incoming client requests

As you can see, the Server class is really small and fairly easy to follow.

Once we complete the implementation of our Server program, we can compile it:

```
% javac Server.java
```

### Arithmetic Client

Now we are ready to implement a client program that uses the services offered by the server.

#### Develop the Client Program

Most of the files we use in implementing the client program are actually contained in the `Arith` package generated by the `idl2java` compiler. The `Client` class, shown in Listing 6, implements the client application that obtains the sum of two arrays. The `Client` class performs the following:

- Initializes the ORB
- Binds to an `Add`
- Requests the `Add` to sum two arrays with the specified values
- Gets the sum of the two arrays using the object reference returned by the `sum_arrays` method
- Prints out the sum of the two arrays

This brings us to the final step in developing a CORBA application using VisiBroker for Java.

### Starting the Smart Agent, Server and Client

The VisiBroker Smart Agent, `osagent`, provides a fault-tolerant object location service and provides runtime licensing of VisiBroker applications. We start the `osagent` on a UNIX system, as follows:

```
% osagent &
```

Once the `osagent` is running, we can start the server program, which can be started as a normal standalone Java application:

```
% vbj Server &  
AddImpl[Server,oid=PersistentId[repId=IDL:Arith/Add:1.0,objectName=Arithmetic Server]]  
is ready.
```

Finally, we can start the client program:



```
% vbj Client
The sum is: 6 6 6 6 6 6 6 6
6 6
```

**Note:** vbj is a korn shell script that sets up the paths, does some initialization and invokes the Java interpreter.

## Running Applications on Different Hosts

In the final step above, we assumed that both the client and server were running on the same host. It is more common, however, to have the client and server run on different hosts. One way to do this is by having an osagent running on each host. However, VisiBroker makes it unnecessary to do so by

providing the OSAGENT\_ADDR property. Thus, to start a client or a server on a host that is not running the osagent, we can use the OSAGENT\_ADDR property to specify the host that is running an osagent. As an example, if there is an osagent running on the host "java100", the following command can be used to start a client on a different host:

```
% java -DOSAGENT_ADDR=java100 Client
```

## Conclusion

In this article, we introduced CORBA and presented the development of a small application using VisiBroker ORB for Java. In future articles I will introduce more advanced tutorials on CORBA. 🍌

## Resources

1. <http://www.omg.org>
2. <http://www.visigenic.com>
3. <http://www.iona.com>
4. <http://www.javasoft.com>

## About the Author

Qusay H. Mahmoud is a Technical Instructor at the Etisalat College of Engineering, United Arab Emirates. Previously he worked as a Senior Software Engineer in the School of Computer Science at Carleton University, Canada and a Software Designer at Newbridge Networks, Canada. He is the author of an upcoming book on distributed programming using Java. Qusay can be contacted at [dejavu@acm.org](mailto:dejavu@acm.org)



[dejavu@acm.org](mailto:dejavu@acm.org)

### Listing 1: Arith.idl

```
module Arith {
    interface Add {
        const unsigned short SIZE = 10;
        typedef long array[SIZE];
        void sum_arrays(in array a, in array b, out array c);
    }
}
```

### Listing 2: % idl2java Arith.idl

```
Creating: Arith
Creating: Arith/AddPackage
Creating: Arith/AddPackage/SIZE.java
Creating: Arith/AddPackage/arrayHolder.java
Creating: Arith/AddPackage/arrayHelper.java
Creating: Arith/Add.java
Creating: Arith/AddHolder.java
Creating: Arith/AddHelper.java
Creating: Arith/_st_Add.java
Creating: Arith/_sk_Add.java
Creating: Arith/_AddImplBase.java
Creating: Arith/AddOperations.java
Creating: Arith/_tie_Add.java
Creating: Arith/_example_Add.java
%
```

### Listing 3: Add.java

```
public interface Add extends org.omg.CORBA.Object {
    public void sum_arrays(
        int[] a;
        int[] b;
        Arith.AddPackage.arrayHolder c
    );
}
```

### Listing 4: AddImpl.java

```
public class AddImpl extends Arith._AddImplBase {
    /** Construct a persistently named object. */
    public AddImpl(java.lang.String name) {
        super(name);
    }
    /** Construct a transient object. */
    public void sum_arrays() {
        super();
    }
    public void sum_arrays(
        int[] a,
        int[] b,
        Arith.AddPackage.arrayHolder c
    ) {
        c.value = new int[10];
        for (int i = 0; i < Arith.AddPackage.SIZE.value; i++) {
```

```
c.value[i] = a[i] + b[i];
        }
    }
}
```

### Listing 5: Server.java

```
import org.omg.CORBA.*;

public class Server {
    public static void main(String argv[]) {
        try {
            // initialize the ORB
            ORB orb = ORB.init();
            // initialize the BOA
            BOA boa = orb.BOA init();
            // create the AddImpl object
            AddImpl arr = new AddImpl("Arithmetic Server");
            // export the newly created object
            boa.obj_is_ready(arr);
            System.out.println(arr + " is ready.");
            // wait for incoming requests
            boa.impl_is_ready();
        } catch (SystemException se) {
            se.printStackTrace();
        }
    }
}
```

### Listing 6: Client.java

```
import org.omg.CORBA.*;

public class Client {
    public static void main(String argv[]) {
        int a[] = {2, 2, 2, 2, 2, 2, 2, 2, 2, 2};
        int b[] = {4, 4, 4, 4, 4, 4, 4, 4, 4, 4};
        Arith.AddPackage.arrayHolder result = new
        Arith.AddPackage.arrayHolder();
        try {
            // initialize the ORB
            ORB orb = ORB.init();
            // Locate an Add object
            Arith.Add add = Arith.AddHelper.bind(orb, "Arithmetic Server");
            add.sum_arrays(a, b, result);
            System.out.print("The sum is: ");
            for (int i = 0; i < Arith.AddPackage.SIZE.value; i++) {
                System.out.println(result.value[i]);
            }
        } catch (SystemException se) {
            se.printStackTrace();
        }
    }
}
```







# Focus: The Java Platform

## An Examination of the Roles Played by Individual APIs that Make up the Java Platform

by Ajit Sagar

Welcome back to the Cosmic Cup. I hope you are enjoying our voyage through the Java universe. Last month we examined the APIs that are formally defined under the scope of the Java Platform for the Enterprise. We're going to change the course of our journey a bit. This month we will look at the APIs that define the Java Platform itself.

### What is the Java Platform?

Before getting deeper into the discussion, I would like to comment on the terms "Java Platform" and the "Java Enterprise." The Java Platform is defined as "*a new operating environment for delivering and running highly interactive, dynamic, distributed and secure applications on network computers.*" It is a layer on top of existing operating systems and hardware platforms that enables the compilation of software programs to bytecodes, which are machine instructions for a virtual machine contained in the Java Platform. The virtual machine is more than an interpreter – it provides additional services beyond the translation of byte codes to native machine code, such as garbage collection, thread synchronization and security management.

On the other hand, the Java Enterprise APIs are Java's interface into business or enterprise computing. They are a category of Java APIs that fall under the definition of the Java Platform APIs; i.e., the Enterprise APIs are a subset of the Java Platform APIs.

### Components of the Java Platform

The Java Platform defines an environment that includes a virtual machine, a programming language, core class libraries and class extension libraries. In short, the Java Platform defines and encompasses *Java*. The greatest benefit of Java is its software platform. This platform creates a virtual operating environment that is capable of producing binary code that can be run vir-

tually on any hardware platform that implements the Java virtual machine. The Java Platform has three basic parts:

- The Java Virtual machine (JVM)
- The Java Language (Language Syntax)
- Java Class Libraries or Java Application Programming Interface (API)

The Java class libraries, which is the interface published by the Java Platform for interacting with the rest of the computing world, comprises the following:

- The Java Core API
- The Java Standard Extension API
- Non-standard Java APIs

The Java Enterprise APIs discussed in last month's article focused on Java for the Enterprise (these are APIs that extend the scope of Java to the world of business computing). The current focus of this column is not on the core and extension APIs that are used for programming Java. *JDJ* already has several columns that have discussed these and several good texts cover them in detail. We're focusing on the APIs that extend Java beyond its core environment. Figure 1 shows the Java platform.

This month we'll take a look at the APIs that are defined under the scope of the Java Platform. Please note that we are, for the moment, foregoing discussion on the Java APIs that focus on Java-related hardware devices, embedded Java and Java operating system APIs (JavaOS). We will also avoid discussions on Java

products (Java Web Server, JavaBlend, etc.) or other APIs (JavaMail, JavaSpaces, etc.). Discussion on these products and APIs is beyond the scope of our current focus. I may cover them in future articles.

### The Java Platform API Categories

The Java Platform APIs are a set of interfaces that are used by developers to build Java applications and applets. All Java Platform APIs are created primarily by JavaSoft with the help of industry-wide specialists in different technologies and market areas.

The Java Platform APIs fall under the following categories:

- Java Base Platform
- Commerce
- Security
- Media
- Enterprise
- Server

These API categories are described in Table 1. The next section briefly examines the Java APIs that fall into each category.

### The Java Platform Constituent APIs

Of the API domains listed in Table 1, we discussed the Enterprise APIs in the last issue of the *JDJ* (Vol. 3, Issue 4). The Java Base API will not be discussed here, as information about it is widely available. The APIs under the other four groups and some related products are listed in Table 2. Subsequent sections of this article briefly describe the individual APIs.

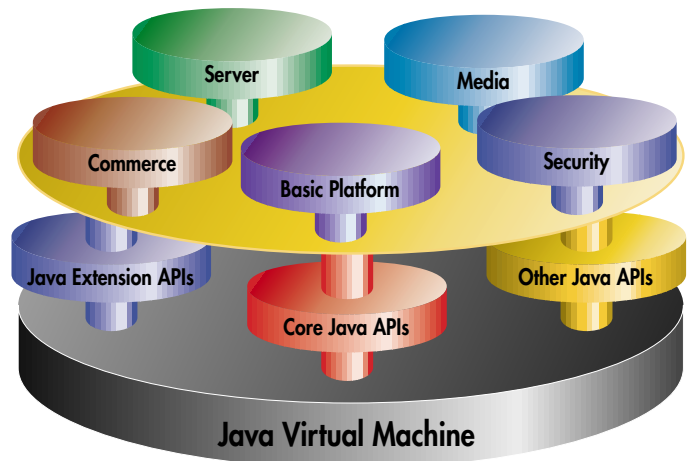


Figure 1: The component APIs of the Java Platform.



## Commerce API (Java Commerce Client)

The *Java Commerce Client (JCC)* is an open, extensible framework, which provides the ability to build electronic commerce applications. JCC offers a Wallet-like user interface, a database and an extensible platform that enables the use of a variety of payment instruments and protocols for E-commerce operations.

JCC provides the API for Java commerce applications. *Java Wallet*, on the other hand, is a family of products that uses the JCC to ensure secure electronic commerce operations. The *Java Wallet* incorporates the JCC, Commerce JavaBeans components, the Gateway Security Model and Java Commerce Messages to offer users an extensible platform for online commerce.

A *Commerce JavaBean component* is a reusable commerce component that meets specific interface requirements to enable development of commerce components that can extend the functionality of the JCC. The Commerce JavaBeans model extends the JavaBeans model to provide interface typing and support for the Gateway Security Model. JavaBeans components show some functionality within standard JavaBeans builder tools, but they only manifest full functionality in Commerce JavaBeans aware environments such as the JCC.

Commerce JavaBeans components are contained in *cassettes*, which package these components. When a cassette is downloaded and installed, the JCC can make use of the Commerce JavaBeans component(s) it contains to perform commerce operations.

The *Java SmartCard API* is the final piece that defines the building blocks for Java's electronic commerce domain. This API enables communication between portable Java applications and smart cards, independent of hardware devices. The *Java Smart Card layer* makes the card reader and port transparent to the Java application communicating with the smart card, providing a direct channel between application and smart card.

The Java Commerce APIs are delivered separately from JDK. They constitute a standard extension to the JDK. The *Java SmartCard API* is currently available as an early access release. The *Java Card API* is available as a 2.0 specification. The *Commerce JavaBeans API* specification has not been published yet.

## Security API

Security in the Java Platform is provided at different levels, via core APIs as well as an extension API. JDK 1.1.x includes *Javakey* and *Jar*, which use the new JDK

## API Category Description

<b>Java Base Platform API</b>	The minimum set of Java APIs that are required to run Java applets and applications. The Java Base API is also known as the Java Applet API. This is distributed with the Java Development Kit and precludes the need for any additional class libraries for writing portable Java programs.
<b>Commerce API</b>	Used to be known as the Java Electronic Commerce Framework (JECF), and is now called the Java Commerce Client. The JCC is Sun Microsystem's Java architecture for secure electronic commerce transactions.
<b>Security API</b>	Provides a framework to ensure privacy and authentication for all Web communications. It is designed to allow developers to incorporate both low and high-level security functionality.
<b>Media APIs</b>	Enables developers to create Java applets and applications that can handle a wide range of rich, interactive media types - including audio, video, 2D imaging, 3D graphics, animation, collaboration, telephony and speech.
<b>Enterprise APIs</b>	Connects Java to enterprise information resources. It enables enterprise developers to build distributed client/server applications in Java that can connect to databases, interact with transaction services and interoperate with existing applications through powerful Java to CORBA interfaces.
<b>Server API</b>	Enables the development of Java-based servers for the Internet and the Intranet. This API provides an extensible framework that contains server-side class libraries for server administration, access control and dynamic server resource handling.

Table 1: The Java Platform API Categories

security APIs provided with the JDK. *Javakey* is used to generate keys and certificates and sign JAR files. *Jar* is a file archiving utility. The security APIs support hashing, digital signatures, and parsing and generating X.509 certificates.

Security support is also included in the client/server products. The *Java Web Server* includes security support for the server-side. The *HotJava Browser* offers fine-grain control for signed applets and built-in SSL support.

The *Java Cryptography Extension (JCE)* provides a framework for encryption and key negotiation. It includes interfaces and implementations of ciphers, secure Java streams, key generation and other features. The current version of JCE, JCE1.2, is designed so that other crypto libraries can be plugged in as service providers and new algorithms can be added. JCE1.2 supplements JDK1.2 (Java Development Kit), which already includes interfaces and implementations of message digests and digital signatures. JCE1.2 requires that you already have JDK1.2 installed.

## Media APIs

The *Java Media APIs* support the integration of audio and video clips, animated presentations, 2D fonts, graphics and images, as well as speech input/output, 3D models and telephony. By providing stan-

dard players and integrating these supporting technologies, the *Java Media APIs* enable developers to distribute compelling, media-rich content.

The *Java 2D API* is a set of classes for advanced 2D graphics and imaging, encompassing line art, text and images in a single comprehensive model. These classes will be provided as additions to packages in the Java 1.2 AWT class libraries.

The *Java 3D API* is a set of classes for writing three-dimensional graphics applications and 3D applets which provide high level constructs for creating and manipulating 3D geometry and for constructing the structures used in rendering that geometry. The early access Java 3D 1.1 Alpha 2 API is available for review.

The *Java Advanced Imaging API* allows sophisticated, high-performance image processing to be incorporated in Java applets and applications. This API will be available as a standard extension to the Java platform. The *Java Advanced Imaging API* implements a set of core image processing capabilities, such as image tiling, regions of interest and deferred execution. The *Java Advanced Imaging API* is being developed by Sun, Autometric, Kodak and Siemens.

The *Java Media Framework API (JMF)* specifies a simple, unified architecture, messaging protocol and programming interface for media players, media capture



and conferencing. JMF is in its first API release, which supports the synchronization, control, processing and presentation of compressed streaming and stored time-media, including video and audio. The JMF 1.0 API was developed by Sun, Silicon Graphics Inc. and Intel Corporation.

*Java Sound* is a very high quality 32 channel audio rendering and MIDI controlled sound synthesis engine with a new Java Sound API. Java Sound will be distributed in two phases. The first phase will be as an engine that will be a core library in JDK1.2. The second phase will be the release of a full featured Java Sound API.

*The Java Speech API* enables incorporation of speech technology into user interfaces of Java applications. It has a cross-platform API to support command and control recognizers, dictation systems and speech synthesizers. The beta Java Speech API is available now for review. The Java Speech API was developed by Sun, AT&T, Dragon Systems, IBM, Novell and Texas Instruments. Java Speech API is in 0.06 beta Specification.

*The Java Telephony API (JTAPI)* is a set of modular, abstract interface specifications for computer-telephony integrated call control. This specification was developed by Sun, Dialogic, IBM, Intel, Lucent, Nortel, Novell and Siemens. The current version of JTAPI is 1.2.

### Server API

*The Server API* enables development of server-side applications in Java. Server-side development is done using *Servlets*, which are Java's server-side counterparts for applets. The Servlet API is an extension to the standard JDK. The Java Servlet Development Kit (JSDK) is used to program Java Servlets. JSDK includes a servlet engine for running and testing servlets, and support for Netscape, Microsoft and Apache Web servers.

*The Java Web Server (JWS)* is a product offered by Sun Microsystems that supports Servlets. JWS is used for developing network servers in the Java programming language.

*The JavaServer Engine* is a collection of reusable Java classes that automates connection management, security and administration. It's used for the development and deployment of network enabled, server-based applications.

### Conclusion

In this article we examined the Java Platform and its constituent APIs. We briefly examined the roles played by individual APIs that make up the Java Platform. Links for detailed information on all these APIs may be obtained from Sun's Java Website at <http://java.sun.com/products>.

## Commerce API (JCC)

API	Full Name	Role
Java Smart Card API	Java SmartCard API	Implements APIs that enable smart card transactions via card reader terminals attached to the Java Wallet user's computer.
Java Commerce JavaBeans API	Java Commerce JavaBeans API	Allows development of Commerce JavaBeans components, which are reusable components that meet specific interface requirements

## Security API

API	Full Name	Role
JCE API	Java Cryptography Extension API	The Java Cryptography Extension (JCE) provides a framework for encryption and key negotiation.

## Media and Communication APIs

API	Full Name	Role
Java 2D API	Java 2D API	A set of classes for advanced 2D graphics and imaging, encompassing line art, text, and images.
Java 3D API	Java 3D API	A set of classes for writing 3-dimensional graphics applications and 3D applets.
Java Advanced Imaging API	Java Advanced Imaging API	Allows sophisticated, high-performance image processing to be incorporated in Java applets and applications.
JMF	Java Media Framework API	Specifies a unified architecture, messaging protocol, and programming interface for media players, media capture, and conferencing.
Java Speech API	Java Speech API	Allows Java applications and applets to incorporate speech technology into their user interfaces
Java Sound API	Java Sound API	A 32 channel audio rendering and MIDI controlled sound synthesis engine with a new Java audio API.
JTAPI	Java Telephony API	A portable, object-oriented application programming interface for Java-based computer-telephony applications.

## Server API

Servlet API	Java Servlet API	Allows server-side Java programs to run with any major Web server and thus supports the middleware layers of the enterprise.
-------------	------------------	--

Table 2: The Java Platform APIs

## Cosmic Reflections

The APIs we discussed in the previous month, and this one, do not in any way, complete the list of APIs associated with Java. And even amongst these APIs, most are in various stages of evolution. Furthermore, new APIs are being defined every month by several different corporations in collaboration with Sun Microsystems. The growing complexity of the Java Platform is making it increasingly difficult for the average developer to comprehend the world of Java. While "simple" is one of the first white paper buzzwords that was used to define

the Java programming language, the Java Platform is proving to be paradoxical to the notion of simplicity. ☛

### About the Author

Ajit Sagar is a member of the Technical Staff at i2 Technologies, Dallas, TX. He holds a B.S. in Electrical Engineering from BITS, Pilani, India and an M.S. in Computer Science from Mississippi State University. Ajit focuses on UI, networking and middleware architecture development. He has 7 and 1/2 years of programming experience, two in Java.



Ajit\_Sagar@i2.com





# Dealing with Network Timeouts in Java

*Using a timer thread or socket options*

by David Reilly

When developing Java network applications in a stable and controlled environment, it's easy to become complacent and ignore the possibility of network timeouts. After all, with the perfect client and server running over a local area network, timeouts won't occur to stall your application. But when your users run clients or servers over the Internet (an environment where networks can go down, badly written software can stall or communication sessions can deviate from the ideal path of a communications protocol), timeouts can cause problems if there isn't a mechanism to recognize and deal with it.

In this article, I'll present two approaches to dealing with network timeouts. The first approach, writing a multi-threaded application, is backwards compatible with JDK1.02, at the expense of increased complexity. The second approach is by far the easiest. In most cases it involves adding only a few lines of code to your application but requires a JDK1.1 virtual machine. As an example, I'll show how the two approaches can be applied to a simple finger client.

## Multi-threaded Applications

Designing a multi-threaded client or server offers a solution to the problem of timeouts. When one thread becomes blocked, waiting for network input or to send more data, a second timer thread can still perform other operations when a timeout occurs, such as terminating the connection. If you've never written a multi-threaded application before, consult the sidebar "Overview of Threads."

Listing 1 introduces the Timer class which is a subclass of `java.lang.Thread`. When an application creates an instance of Timer, it specifies the number of milliseconds that can elapse before a network timeout occurs. Once the timer is started, it will begin to decrement an internal counter. When the counter has no more milliseconds remaining, the timeout action occurs

which, by default, is to exit the application. If necessary, subclasses of Timer can override the timeout method to provide custom functionality.

For the sample finger client, we will use the Timer class to terminate the application if a network timeout occurs (for example, a server that stalls). Listing 2 shows a multi-threaded finger client and Listing 3 shows a sample finger server. The client and server will communicate via port 79 (defined in the public int FINGER\_PORT). However, on some systems the server may be unable to bind to this port, in which case an ephemeral port in the range 1024 to 5000 should be used instead for both client and server.

The finger server reads data from files matching the format 'user-name.info' where name is a valid username. For testing purposes, you can create a single user file and have the finger client request information. To simulate timeouts, the server maintains a counter and stalls after the first line of input on every second connection.

The finger client starts by checking its command line arguments and creating a new instance of itself. The client then opens a socket connection and gets input and output streams. It sends the name of a user to the finger server and then begins reading in data.

At some point in the connection, our finger server will stop sending data and our client will be stalled while it waits for input. Without some means of reconciliation, our client would remain blocked – this is where our Timer class solves the problem.

It is the responsibility of the

application to repeatedly reset the timer to prevent a "false" timeout from occurring. While it is looping, the timer is reset; if the loop stops for some reason, the timer can continue uninterrupted and generate a timeout. When this occurs, the program can terminate rather than being locked.

If we wanted our timer to have different properties (for example, re-establishing a connection), we could simply extend the timer to create a custom timer class and to override the timeout method. Were we to write a server that supported timeouts, we would need a different timer thread for every single connection (which could cause considerable overheads if a large number of concurrent connections were generated). A much more efficient method is to use the new socket features introduced in JDK1.1, as we'll see in the next section.

## Overview of Threads

Applications typically have many tasks to perform. In some situations these tasks can be executed sequentially. However, this means that every other task must remain idle until the current task is performed. In networking applications, if your program is reading input from a communications channel and becomes blocked waiting for input, your application will remain in that state until such time as data arrives or the connection is terminated. This isn't a desirable outcome because the input may never actually arrive and the program may become unresponsive to the user.

Some applications, however, have multiple threads of execution. This means that while one part of the program is perhaps reading data, another can be performing other useful tasks. Even in single CPU systems, multiple threads of execution can be running. (Only one thread will be active at a time, but threads are allocated small portions of CPU time so the appearance is given that multiple tasks are being performed concurrently.)

Use of threads under Java is a large topic itself – there are plenty of books and articles available that cover this area. For further information, you may want to check out the book "Java Threads" by Oaks and Wong, published by O'Reilly & Associates.



## Timeouts Made Easy

While the multi-threaded approach works well and is backwards compatible with JDK1.02, it does introduce a great deal of complexity, even for simple applications such as a finger client. If it is important to run your application on older virtual machines, use multi-threading. However, if you have the luxury of a JDK1.1 platform, you can add timeout code to your application with ease.

As part of the new features of JDK1.1, support for socket options have been introduced. One of these allows developers to specify the number of milliseconds that must elapse before a timeout occurs, throwing a `java.io.InterruptedIOException`. For Socket connections, this means that operations reading from the socket input stream can timeout if no data is received within a given period of time. With only a few lines of code, we can introduce support for timeouts, without moving to a multi-threaded design in our clients and without launching a Timer thread for our servers.

## Simple Finger Client

Listing 4 shows a simple finger client which can be run against the server from Listing 3. You'll notice how simple it is to set a timeout value. The `setSoTimeout` method accepts an integer value as a parameter, representing the number of milliseconds for a timeout.

```
// Set SO_TIMEOUT for five seconds
socket.setSoTimeout ( 5000 );
```

All you need to do now is catch the `java.io.InterruptedIOException` that is thrown when a timeout occurs while reading from a Socket's input stream and your application will handle timeouts.

## Timeouts in Servers

Timeouts can occur in servers as well. If you're writing a multi-threaded server, it's important to set and detect timeouts in the socket connections to clients. Otherwise, badly behaved clients can use up memory by stalling and leaving threads open, or use up all your available connections if your server has a limit to the number of connections it supports. A good design would be to call the `setSoTimeout` method on every socket you accept and to catch `InterruptedIOExceptions` if thrown.

The most important case for using the `setSoTimeout` method is when your server uses `DatagramSockets`. `DatagramSockets` allow Java applications and applets to communicate via UDP, which doesn't offer guaranteed packet delivery or ordering. It's critical that applications using UDP recognize

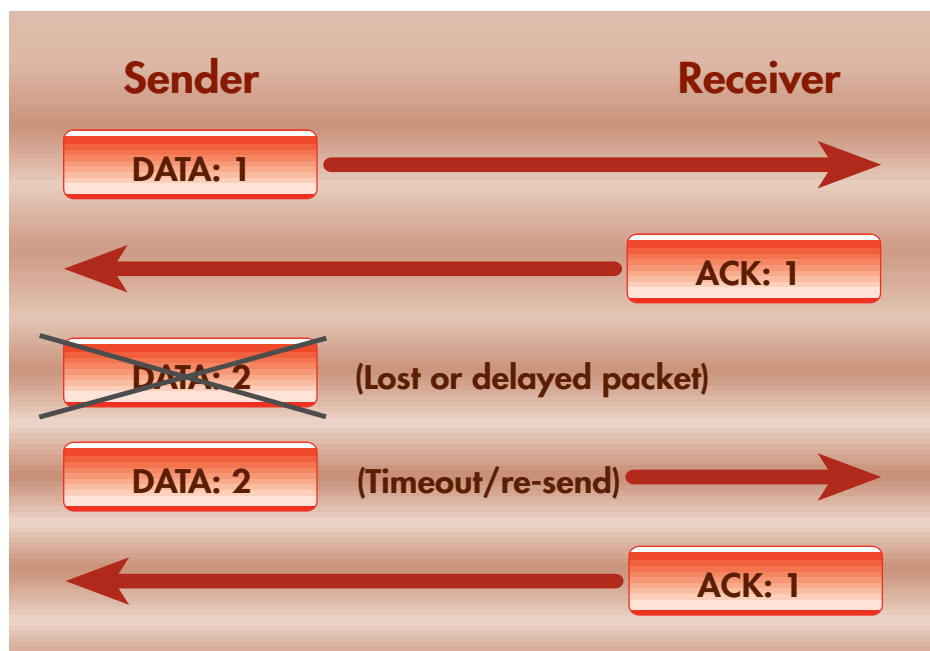


Figure 1: Lost packet causes timeout

and provide for situations in which timeouts occur. Previously, this meant creating a separate thread to monitor packet arrivals (and often re-sending packets when no response was issued). Now, however, your application can easily cater to this situation with a few lines of code using the `setSoTimeout` method.

To illustrate this, I've written an example program that implements a stop-and-wait delivery mechanism. One application (the sender) sends small packets of data containing a sequence number and then waits for an acknowledgement packet to be sent by the receiver. If no acknowledgement arrives, then one of two situations could have occurred:

1. The original packet never arrived at the receiver's end.
2. The original packet arrived and was acknowledged, but the acknowledgement never arrived at the sender.

Listing 5 shows the code for the sender and Listing 6 shows the code for the receiver. The sender uses a `DatagramSocket` to send packets, and receive acknowledgements. Likewise, the receiver uses a `DatagramSocket` to receive packets and send acknowledgements. Both sender and receiver should check for timeouts and respond accordingly.

Timeouts can occur for a variety of reasons; there may be times when the sender is forced to send more than one instance of a data packet, or when the receiver sends more than one acknowledgement. It is important in this situation for both sender and receiver to check the sequence number and handle inappropriate packets gracefully. It is even more important when your

application sends many packets and expects the receiver to maintain sequence integrity.

## Conclusion

The decision to use a timer thread, or to use socket options instead, is largely up to the individual circumstances of the application. Some applications will require backwards compatibility with JDK1.02, in which case timers may be the only option. Wherever possible though, due consideration should be given to setting socket options for timeouts.

Setting socket options for timeouts and catching `InterruptedIOExceptions` allows developers to place code close to the point at which input and output takes place. This reduces complexity and saves placing control of your connections in an external class. For clients that are not already multi-threaded, setting a socket option timeout is by far the better choice and with only a few lines your application can support network timeouts. ☛

### About the Author

David Reilly has worked on network protocols and Web-related programming at Bond University, Australia. Since his conversion to Java in 1996, he has worked almost exclusively with the language, finding it both a joy to use and the most productive way to produce portable applications. David can be contacted at [java@davidreilly.com](mailto:java@davidreilly.com)



[java@davidreilly.com](mailto:java@davidreilly.com)

**Don't Type it... Download it!**  
Access the source code for this and other articles appearing in this issue at [JavaDevelopersJournal.com](http://JavaDevelopersJournal.com)

# Static Initializers and Uninitializers



## A discussion of class loading and unloading

by Brian Maso

I haven't found a good discussion on the topic of class loading and unloading in my searches through Java literature or Java resources on the Web, so I thought it would be a good topic to cover this month. This month's column is all about how and when Java classes are loaded and unloaded, and how you can write classes that link into that process.

Let's start off with class loading. Java classes are loaded using objects of type `java.lang.ClassLoader`. Each VM has exactly one default class loader called the system class loader. Basically, when the VM detects that a particular class needs to be loaded, say a class named "Foo", the VM asks its system class loader to find "Foo". It's the class loader's job to find some resource that defines that class. It is not of the VM's concern whether that resource exists in a .CLASS file on the local file system, in a .CLASS file accessed through an HTTP server over the Internet, in a .ZIP file (or in some other location) or is even generated on-the-fly by the class loader itself. That's the class loader's concern. It's the class loader's job to find that resource somehow and load it using the method `ClassLoader.defineClass()`.

Most system class loaders work pretty much the same way. The system class loader is aware of a system property called the classpath. The classpath is a list of local file system directories and/or .ZIP files. (And, starting with Java 1.1, it might also include .JAR files, which are a lot like .ZIP files.) Conceptually, this classpath is a list of locations to find .CLASS files.

When asked to load class "Foo", a system class loader will create the file name "Foo.class" by just sticking the extension ".class" onto the class name. The system class loader will search for this file using the entries in the classpath. For example, if the classpath contains the list of entries "A;B;C", three directory names, the system class loader will first try to find the file with the full path "A\Foo.class". Failing that, it will

then look for "B\Foo.class". And finally, failing that, it will look for the file "C\Foo.class". If that third step fails, the system class loader will throw an exception of type `ClassNotFoundException`.

Assuming the system class loader finds the file "Foo.class" in one of the directories on the classpath, it will load the resource into the VM using `ClassLoader.defineClass()`. This is where things start to get interesting. The `defineClass()` method does several steps first to ensure the .CLASS resource is in the right format and the code in the class will not harm the VM (a process known as "verification"). Once that is done, `defineClass()` will create a new object of type `java.lang.Class` to represent the newly loaded Java class. This new object is known as the "Class object".

The Class object represents the loaded class to Java code. For example, to access the Class object of any object, you can call the object's `getClass()` method:

```
// Accessing the Class object of Object "o"
Class cls = o.getClass();
String classname = cls.getName();
```

Getting back to class loading, Java classes can have dependencies on each other. This means that in order to load class A, it is also required to load class B. In this case, we would say class A is dependent on class B. Class dependencies are created whenever you explicitly refer to class B in class A's code. For example:

```
// Class Foo is dependent on class Bar
public class Foo {
    public void xyz() {
        Bar barObject = new Bar(); // explicit
        dependency
    }
    ...
}
```

The explicit dependency of class Foo on class Bar means that the class Bar will automatically be loaded when it is required to be used by class Foo. These dependencies between classes are automatically detected and made to load required classes when they are needed. That is, when the method `xyz()` is called in class Foo, the Bar class will automatically be loaded and a corresponding Class object created.

There are several ways an explicit dependency can be set up:

- Defining a class variable or local variable of type Bar in Foo creates a dependency of Foo on Bar.
- Declaring a return value of type Bar in a method of class Foo creates a dependency of Foo on Bar.
- Deriving Foo directly from class Bar creates a dependency of Foo on Bar.

Note that there is one other step that is completed before the Class object is returned from `defineClass()`. That is the "static initialization" step and it happens as the last step of class loading. Each class has a static initializer block. This is a block of code that is to be run as soon as the class is loaded – before any objects of the class are created or any methods of the class called. The purpose of the static initializer block is to initialize the static members of the class and to do any other special initialization required for the class to work. Other special initializations include loading dynamic libraries that implement native code, making network or database connections that might be required, etc. Take a look at the three classes in Listing 1, which demonstrate these static initializer blocks. Both classes Foo and Bar have static initializer blocks. Class Foo is dependent on class Bar. The Main class implements a `main()` method (a stand-alone application), and Main is dependent on Foo because Foo is mentioned as a variable type in class Main. The static initializer blocks just print out a line of text to `System.out`. This is the output you would see if you ran Main using the JDK 1.1 interpreter:

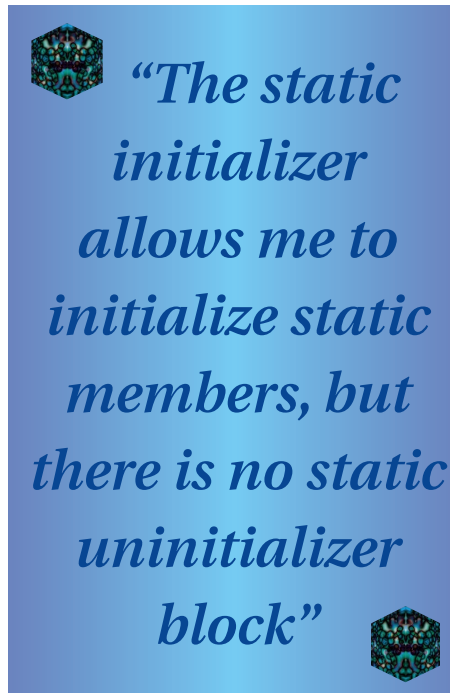
```
Loading class Main
Starting the program
Loading class Foo
Created the Foo object
Loading class Bar
```

Starting with Java 1.1, classes could be unloaded just like they could be loaded. The rule for unloading a class is surprisingly simple and makes sense. When the Class object for a class is garbage collected, the class is automatically unloaded. Use of the garbage collector and the Class objects to represent the loaded state of a class is very elegant, understandable and intuitive.

**Note:** When an explicit dependency is set up between two classes, the required class will not be unloaded until the dependent class is unloaded. There is an explicit relationship set up between the Class objects. That is, as long as the Class object for class Foo sticks around (and the Foo class is thus loaded into memory), the Class object for Bar will stick around.

It is possible to set up “implicit” dependencies between classes. An implicit dependency means that the VM is not aware of the dependency. Instead, your program takes control of loading classes when they are required. To force the system class loader to load a class, you use the static method `Class.forName (String classname)`. Listing 2 shows a simple class Foo which loads the class Bar using code, instead of using an explicit reference to the Bar class. This prevents an explicit relationship between Foo and Bar. In this case, it is possible for the class Bar to be loaded and unloaded, and even reloaded, several times during the life of the VM. The program in Listing 2 attempts to force the VM to garbage collect the Bar class Class object by filling memory with orphan objects every time the ENTER key is pressed by the user. Sample output from this program running in the JDK 1.1 is:

```
***Loading Foo class
Starting the application
***Loading Bar class
Filling memory with 1000 useless objects
<Enter key pressed by user>
Filling memory with 1000 useless objects
<Enter key pressed by user>
***Loading Bar class
```



Filling memory with 1000 useless objects  
...

Note that the Bar class static initializer was called twice. That is, the Bar class was unloaded at some point and then reloaded when it was later required.

The fact that, starting with Java 1.1, classes could be unloaded, leads to an important feature actually missing from Java. While it is easy for me to write code that is called when a class is loaded (the static initializer block), there is no companion block called when a class is unloaded. The static initializer allows me to initialize static members, but there is no static uninitializer block that allows me to clean up resources.

When would such a static uninitializer be useful? Imagine I've created a static-only class that allows me to allocate and deallocate chunks of native memory. The public interface for this NativeMemory class might look something like this:

```
public class NativeMemory {
    public static int alloc(int size); //
    returns handle
    public static void setmem(int handle,
    byte[]);
    public static void dealloc(int handle);
    public static int realloc(int handle, int
    newsize);
    ...
}
```

For such a NativeMemory class, a static uninitializer would be crucial. Without it, there would be no way for the class to release allocated native memory that the client code forgot to deallocate.

It would be great if you could provide a `finalize()` method to be used by the NativeMemory Class object. The class is unloaded when the garbage collector finalizes the Class object. This would be the perfect time for the NativeMemory class to release any native memory that hasn't been released yet. The `finalize()` method would, in effect, become the static uninitializer for the class.

Unfortunately, it is not possible to change the `finalize()` method implementation for the Class class. Instead, however, you can delegate the responsibility of class uninitialization to another object. A reference to that object would be stored in a static variable in the NativeMemory class. This means that the clean-up object would only be garbage-collected when the NativeMemory Class object was being collected. That is, when the NativeMemory class was being unloaded. The `finalize()` method of the delegate object would then become the static uninitializer for the NativeMemory class.

Listing 3 shows how a NativeMemory class could use a static reference to a delegate object to manage its static class uninitialization. The delegate object would, ideally, be a static inner class object of the NativeMemory class which would allow the delegate access to the private static member of the NativeMemory class.

Note that, like object finalization, class

### Listing 1: Class dependency. Main is dependent on Foo, is dependent on Bar.

```
public class Main {
    public static void main(String[] astrArgs) {
        System.out.println("Starting the program");
        Foo f = new Foo();
        System.out.println("Created the Foo object");
        f.xyz();
    }

    // static initializer
    static {
        System.out.println("Loading class Main");
    }
}
```

```
class Foo {
    static {
        System.out.println("Loading class Foo");
    }

    public void xyz() {
        Bar bar = new Bar();
    }
}

class Bar {
    static {
        System.out.println("Loading class Bar");
    }
}
```

unloading is not guaranteed to occur at any particular time. Better VM implementations will unload classes that haven't been used recently and that are no longer being referenced. Worse implementations may never unload unused classes. The best you can do is to write your classes with static uninitializers when required, just in case the VM does a good job unloading your class.

So, the point of this column has been threefold:

- To give a primer on class loading for those Java programmers who might not have a good idea of how it works

- To explain how explicit and implicit class relationships are important
- And something for those who already know all about class loading: How to make static uninitializers for your classes

**One final note:** If your class really requires static uninitialization, even if the application quits because `System.exit()` is called, there is a new facility in Java 1.1 that ensures object clean up on exit. Take a look at the new `System.runFinalizersOnExit()` method. ☛

### About the Author

Brian Maso is President of Blumenfeld & Maso, Inc., a Java and Media consulting company. In addition to his involvement in several commercial Java projects, he is also the author of several Java books, including Osborne/McGraw-Hill's upcoming title "Visual J++ From the Ground Up". Brian is the Java guru of DevelopMentor's Java training courses. He has also written several Java white-papers for individual corporations. Brian can be reached via e-mail at [bmaso@develop.com](mailto:bmaso@develop.com).



[bmaso@develop.com](mailto:bmaso@develop.com)

### Listing 2.

The implicit dependency of class `Foo` on class `Bar` is caused by the fact that the `Foo` code uses `Class.forName()` to load class `Bar`. This means the `Bar` class may be unloaded and reloaded several times during the lifetime of the `Foo` class.

```
public class Main {
    static Foo f = new Foo();

    public static void main(String[] astrArgs) {
        System.out.println("Starting the application");

        Thread t = new Thread(f);
        t.start();

        try {
            while(true) {
                synchronized(f) {
                    System.in.read();
                    f.notify();
                }
            }
        } catch (Exception e) {
            // poor exception handling,
            // for the sake of brevity
        }
    }
}

class Foo implements Runnable {
    static {
        System.out.println(
            "Loading Foo class");
    }

    public Foo() {}

    public synchronized void run() {
        while(true) {
            System.out.println(
                "Filling memory with " +
                "1000 useless objects");

            Class cls = Class.forName("Bar");
            for(int ii=0 ; ii<1000 ; ii++)
                cls.newInstance();

            cls = null;
            System.gc();

            try {
                wait();
            }
        }
    }
}
```

```
        } catch (InterruptedException ie) {
        }
    }
}

class Bar {
    static {
        System.out.println(
            "****Loading Bar class");
    }

    public Bar() {}
}
```

### Listing 3: A static-only `NativeMemory` class requires a static uninitializer. The static inner class object fulfills that function.

```
public class NativeMemory {
    private static uninitier = new Uninit();
    ...

    // Static initializer loads native library,
    // starts allocating chunks of memory, etc.
    static {
        ...
    }

    // Public, static only interface
    public static int alloc(int size) { // returns handle }
    public static void setmem(int handle, byte[] {...}
    public static void dealloc(int handle) {...}
    public static int realloc(int handle, int newsize) {...}
    ...

    // Private static inner class, a singleton class,
    // the singleton object acts as a class uninitializer
    // for this class
    private static class Uninit {
        public Uninit() {}

        // Called only when NativeMemory class
        // ready to be unloaded.
        public void finalize() {
            // Has access to private static members of the
            // NativeMemory class. Uses them to
            // uninitialize the class.
        }
    }
}
```



# JViews

## by ILOG, Inc.

*This 100% Pure Java Class Library for developing graphics applications could be just what the doctor ordered*

by Ed Zebrowski



Once upon a time, it looked as though I was set for a while on software. I had L-View, the new picture editing shareware everyone was talking about. I had Microsoft Paint, a pretty good graphics creation program (if you didn't mind your graphics looking like a child in first grade had drawn them). I had HTML Notepad and a full head of hair. What more could a budding young Web developer ask for to guarantee success in this brand new industry?

As we all know, a lot has changed since then. The passing of time, as well as the dawning of new technologies, has led to the obsolescence of all of the aforementioned. (It's my hair I miss the most!) By the time the Java revolution exploded, the deck had been re-shuffled and graphics development was no exception to this rule. I've always tried to keep up with the latest developments in graphics, so I was anxious to try JViews, a new Java graphics development class library from ILOG.

I've seen Java graphics packages before, many of which were not very impressive. They offered nothing more than a sparse menu of simple shapes, forms and limited choices of ways the graphics can interact with objects. JViews is different, however.

### Installation

Your system must be enabled with JDK 1.1 or better. A classpath variable has to be set to the directory the install program resides in. Open a command prompt and type: java JViews. Installation is automatic from there.

### It Inherits its Power from C++

ILOG took what they had learned from developing their C++ graphics library and applied it to Java. The result is a library of Java classes that can be used in the development of high performance graphics displays. It is especially useful for network layout displays, map displays (even if the end user wants to overlay these maps with their own customized objects) and customized editors. The displays that can be built with JViews can contain a very large number of graphics, upwards of tens of thousands of objects. This allows JViews to shine for those industrial-strength apps. Another benefit is the separation of presentation from behavior. While one set of pre-defined objects are graphics that know how to draw themselves, another set represents behaviors, such as selection, drag/drop and resizing. This separation has a few big advantages. Developers can assign any behavior to a graphic object to change its function. Pre-defined behavior objects can be subclassed so end users can fine-tune it to their exact needs.

Although JViews is based on C++, it has the completely open architecture that is the cornerstone of Java development. Any Java lightweight component can be taken and used as a node on a network. Some of the JViews classes can even be wrapped as beans and placed on a GUI builder palette alongside other beans. JViews is built in two levels: The 2D graphics level, and the manager level.

### Create Basic or Complex Graphics with JViews' Huge Library

The 2D graphics level provides graphic objects to control appearance, and interaction objects to control behavior. Provided also are the basic tools needed to combine graphic and interaction objects to form a

### JViews

ILOG, Inc.

1901 Landings Dr

Mountain View, CA 94043

Phone: 800-FOR-ILOG

Fax: 415-390-0946

Web: <http://www.ilog.com>

Email: [info@ilog.com](mailto:info@ilog.com)

Price: \$6500 single developer license each

complete application.

The manager level of JViews organizes sets of graphic objects into multiple views and layers with higher order interactions. All of these aspects are grouped together by an object called a manager. The grapher, a class that organizes certain objects into nodes and links, is included among the manager class hierarchy.

JViews comes equipped with *IlvEdit*, a handy editor that is provided with source code. It allows you to easily create and edit graphic objects. The starting point for these objects is the class *IlvGraphic*. This class allows the graphic object to draw itself at any given destination port. If necessary, an associated object of the *IlvTransformer* class may be used to change the coordinates of the graphic object. *IlvGraphic* also has member functions that allow you to set and change geometric dimensions. Various member functions are provided to implement user properties that can be associated with an object for application-specific purposes.

### It Doesn't Just Create Graphics - It Interfaces them as Well

I remember when I knew of only one way to interface graphics with some sort of function or application, and that was through HTML. The HREF tag was used to create a simple link to another Web page or an application from the server that could run through the browser. Today's fast-paced, competitive environment does not allow us to get off so easily, however. There is a very real and immediate need to create GUI interfaced applications that function both on and off the Web. JViews provides



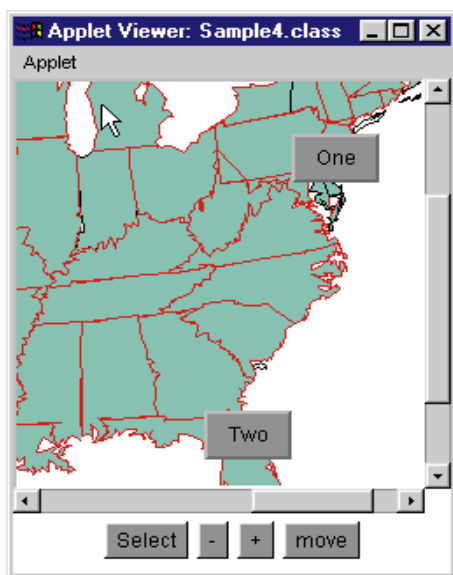


Figure 1: The first example creates a scrolling map of the USA with two custom objects atop it

us with just the tool we need to create these applications.

An application-programming interface (API)—a library of Java classes which contains pre-defined graphic objects and various behaviors which can be applied to these objects—is also included. The result is an easy, no-nonsense approach to interfacing graphic objects to applications. A friend of mine who is in the process of interfacing all of his household lighting through his computer was so impressed with this software that he wants to try to use it to make a GUI interface for his system. Turning on a light or the stereo was never this exciting!

## A Basic Example

Let's take a quick look at a simple example of how JViews is used. First, we'll make a simple map. The first example shows the creation of a manager and how to load a

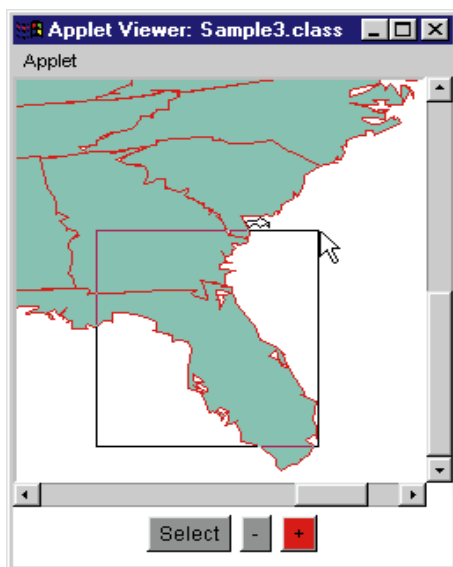


Figure 2: Here, various states can be moved and morphed by clicking and dragging

graphic objects file into this manager. The example given shows a scrolling window of a map of the USA.

Here's some of the code used to create this applet:

First, the library must be imported:

```
import ilog.views.*;
```

The applet and AWT packages must also be imported:

```
import java.applet.*;
import java.awt.*;
```

If the file is going to be loaded through a URL, the java.net package must be used:

```
import java.net.*;
```

The applet class named sample1 is now created. This applet has two fields: The manager (class IlvManager) that stores the objects and the viewer (class IlvManagerView), which displays the content of the manager.

```
public class Sample1 extends Applet
{
    IlvManager manager;
    IlvManagerView mgrview;
    ...
}
```

For the initialization of the applet, the manager is created:

```
public void init() {
    manager = new IlvManager();
    IlvReliefLabel obj = new
    IlvReliefLabel(new IlvRect(580, 160, 50, 30),
    "One";
    manager.addObject(obj, 1, false);
    obj = new IlvReliefLabel(new IlvRect(530,
    330, 50, 30) "Two"
    manager.addObject(obj, 1, false);
    ...
}
```

The second example provided, sample2.java, shows the use of selection interactor. In this example, a button has been added to the applet. Once the button has been clicked on, graphic objects can be selected, moved and it is even possible to modify their shape.

A field named selectInteractor has been added in the class:

```
void createInteractorButtons()
{
    Button button;
    button = new Button("Select");
    button.addActionListener(new Action-
```

```
Listener() {
    public void actionPerformed(Action-
    Event evt) {
        if (selectInteractor == null)
            selectInteractor = new IlvSe-
            lectInteractor();
        if (mgrview.getInteractor() !=
            selectInteractor)
            mgrview.pushInteractor(selectInteractor);
    }
});
add("South", button);
}
```

It might be worth noting that the SelectInteractor is a pre-defined behavior that controls the selection of an object, the "resize handles" that indicate that an object is selected, moved and resized, etc. Developers can choose to use this behavior "as is", or subclass to modify just those parts of it that they want to tailor.

## Other Features

- **Compatibility:** JViews works with all AWT (abstract window toolkit), JFC (Java foundation classes) and JavaBean components. Other Java GUI builders such as Symantec's Visual Café and Sun's Java Workshop can encapsulate JViews classes as JavaBeans to place on the palette and use like other components.
- **Browser support:** JViews can be used on any browser that supports the latest JDK. This includes Netscape Communicator, Sun's HotJava and, of course, Microsoft Internet Explorer.
- **Importation of external formats:** JViews is equipped with file reformatters to import existing 2D graphics data files. The DXF file reader imports AutoCAD files.

All in all, JViews is an excellent library from which really cool apps can be built. It enables developers to quickly and easily produce clean, crisp graphics that can be used for demonstration purposes or to be interfaced with more complex applications. It offers compatibility with other applications and, best of all, it's Pure Java, which means total platform independence. If you are in a position where you need a no-nonsense, hard hitting application to develop GUI interfaced graphics, give it a try. I did, and I loved it. Now if I could just find a way to get it to grow hair! ☘

### About the Author

Edward Zebrowski is a technical writer based in the Orlando, FL area. Ed runs his own Web development company, ZebraWeb, and can be reached on the net at zebra@rock-n-roll.com



zebra@rock-n-roll.com



# Java and the Law

## *Protecting inventive Java development*

by Michael Zarrabian

As far back as the first time two computers were physically linked to one another, the idea of sharing information between, and executing programs on, networked computers has been around. Distributed computer architecture challenged software manufacturers to provide operating systems, communication protocols, computer languages and operating environments in which the distributed nature of the underlying architecture was substantially transparent to developers and users. Now, the Internet generally, and the World Wide Web ("WWW") specifically, have challenged the software manufacturers to again provide such a transparent operating environment for the Internet.

Currently, users access information stored on the WWW using browsers. The Java programming language and Java applets allow development of platform-independent application programs executing over the Internet and the WWW. Therefore, any computer platform with a browser having a Java interpreter can execute Java application programs downloaded from the WWW. As such, existing browsers allow end users browsing HTML files to use Java applets and image files. But why stop there? Why not have a programming environment for the Internet that users and developers can employ to assemble programs such as Java applets from modules distributed locally and over the Internet, or to download entire program folders from different Internet nodes?

Such programming environment technology exists and at least one company has obtained a patent for it. On January 6, 1998, Sun Microsystems, Inc. obtained U.S. Patent No. 5,706,502 ("the '502 patent") directed to Internet-Enabled Portfolio Manager System (PMS). According to the '502 patent, the PMS allows users to manage,

create, edit, debug and compile software portfolios including software projects such as Java applets, standalone executable programs, image files, Java class libraries or remote Java applets. The software portfolios and their contents can be stored on the system hosting the PMS or on any remote system that can be accessed via the Internet using standard Internet communication protocols, such as FTP or HTTP. The PMS includes portfolio files, each of which includes links to the projects that compose

*"Developers or companies should carefully review the '502 patent and seek competent legal advice to ensure their systems do not infringe the '502 patent"*

a portfolio and project files that set out the attributes of one project.

The PMS provides:

1. Portfolio methods that allow users to create, choose, import and remove entire portfolios
2. Project methods that allow users to create, import, choose, edit, remove, run, copy and paste projects

The contents of a particular portfolio or project file determine how the PMS implements such methods. For example, if a user wants to import a portfolio from a remote system, the PMS invokes an integrated Web

browser, which downloads the desired portfolio onto the local system. The PMS also allows users to publish portfolios and projects on the Internet to be used by others within certain limits set by the publisher. For example, the publisher can restrict copying of source programs while allowing copying of executables.

The '502 patent has 35 "Claims" directed to the PMS. Claims are legally operative parts of a patent, granting the patent owner the right to stop others from making, using or selling the invention as specified in each claim. Therefore, in general, if a claim of a patent "reads on" a product (i.e. the product includes every element of the invention as specified in the claim), that product may be infringing the patent. The following is a brief analysis of claim 1 of the '502 patent. The claim is directed to a PMS for portfolios of software projects that are distributed over a set of networked computers connected to the Internet, where the PMS is resident on one of the networked computers.

The elements of the PMS covered by the claim 1 of the '502 patent are:

- A set of portfolio files
- A set of portfolio project files
- A portfolio manager
- A Web browser

Each subset of the portfolio files represents one portfolio with references to a set of project files. Portfolios and their constituent projects can be thought of as directories and files, respectively. The references can be file names for local project files, and URLs for remote project files. Each member of the project files specifies project attributes associated with its portfolio. The portfolio manager includes a set of user-selectable portfolio methods configured to process the portfolios based on information in the portfolio files. The Web browser is configured to download selected remote portfolio files from the Internet to the PMS needed by the portfolio methods. Additionally, the PMS can include a User Interface configured to display portfolios. This includes the projects composing the portfolios in a consistent fashion, independent



of the location of the portfolios and their constituent projects. The portfolio manager User Interface allows user interaction with the portfolio methods. This enables users to determine and manipulate the displayed portfolios using those portfolio methods.

According to the '502 patent, the PMS User Interface is embedded within an application called the Java Workshop (JWS) program. Among other things, this allows users to organize executable programs (Java applets and standalone executables) and non-executable files (image files and Java class libraries) into collections called portfolios. The JWS program has an integrated JWS Browser which allows a user to seamlessly create and work with remote (stored apart from the user's machine or local network) or local portfolios. Furthermore, the JWS Browser allows assembly of local and remote "project" components of a portfolio. The User Interface facilitates user interaction with mixed objects, such as a portfolio consisting of both local and remote projects. It does so by providing a single paradigm for working with all objects regardless of the objects' locations. The differences between working with remote and local objects, such as executing a Java applet stored on a remote computer vs. executing a standalone program stored locally, are handled in the JWS program. The User Interface allows a user to initiate the execution of the remote applet or the local program in the same way (e.g. by double-clicking an icon representing the applet).

The local computer memory contains a set of JWS files that collectively defines the User Interface, methods and data files that compose the JWS. More specifically, the JWS files include the JWS program (hereinafter referred to as the "JWS"), JWS Browser and a group of interface files called the JWS toolbar specification. The JWS toolbar specification is composed of four sub-groups of files: icon specifications, Web documents, JWS applets and other referenced files. These sub-groups of files specify the appearance and, more importantly, the operation of a set of icons that are displayed as elements of a JWS toolbar. The JWS toolbar, which is a key element of the JWS User Interface, is displayed by the JWS program on the JWS window. The JWS window also includes an applet window controlled by JWS applets that are executed by the JWS program in the course of project and/or portfolio management.

Each icon has a corresponding icon specification that defines the icon's visual attributes. It also specifies a link(s) to a Web document(s), listing an initial set of files to be loaded and possibly executed whenever the icon is selected. The links

# What Is a Patent?

by Jonathan D. Farrell and Anthony Coles

A patent is a piece of property. So what can you do with this property? You can keep others off it. Remember when you were a kid and you'd yell "Get off my property!" to another kid? A patent is pretty much the same concept, except that instead of "get off my property", a patent owner can stop others from making, using or selling his invention, and "trespassing" is called "infringement."

## *Does having a patent mean I can make and sell my invention without worrying about infringing someone else's patent?*

Not necessarily. Consider this classic example: A caveman, Og, invents the bucket. He goes to the caveman patent office and is granted a patent. Another caveman, Zug, finds that the bucket is hard to carry when it is full, so he invents a handle for the bucket. He, too, is awarded a patent. Zug can't make, use or sell his bucket with the handle because he infringes Og's bucket patent. Og can't make, use or sell a bucket with a handle without infringing Zug's patent. It may be best if they cross-licensed each other's patents so that they can both make, use and sell buckets with handles.

## *What Is Patentable?*

In the United States, a patent may be granted to "any new and useful process, machine, manufacture or composition of matter, or any new and useful improvement thereof...". Patentable inventions must be novel (new) and non-obvious. "Novel" means it's new and has never been done before. "Non-obvious" is a little trickier to define, but can roughly be described as follows. Imagine that a person of ordinary skill in the technical area of your invention has in his/her lab all of the relevant public knowledge about that technology. Even with all this knowledge available, to that person the invention is more than just a simple combination of that prior knowledge.

## *How Can I Screw Up My Potential Patent Rights?*

There are lots of things that can prevent you from getting a patent, including things you can do yourself. One thing you can do to lose your rights in this country is to wait for more than one year to file a patent application after the invention has been publicly disclosed or offered for sale. (To prevent losing rights in some countries, a patent application should be filed before there is any disclosure of offer for sale). A public disclosure may be a demonstration at a trade show or a journal article describing the invention, for example.

Volumes have been written on all of these topics, so this article is just a basic overview and is not intended to be a detailed treatment of any of the topics. Nor is this article intended to provide legal advice. If you have a question about patent law, you should ask your patent attorney. If you don't have a patent attorney, the U.S. Patent and Trademark Office has a list of all registered patent attorneys. You can call the U.S. Patent and Trademark Office at 800-PTO-9199 or 703-308-HELP.

### **About the Authors**

Jonathan D. Farrell is the corporate counsel for **Java Developer's Journal** and **SYS-CON Publications, Inc.** Anthony Coles is a patent and trademark attorney with the law firm **Meltzer, Lippe, Goldstein, Wolf & Schlissel, P.C.** Both Farrell and Coles can be reached by phone at 516 747-0300.

can be to Web documents stored on the local system (e.g. the local computer), and include a local path and file name that can be handled by the file service provided by the local operating system. The links can also be to remote Web documents (e.g., documents stored on remote computers) that can be retrieved over the Internet by a conventional Web browser. Because the JWS program incorporates the JWS Browser, providing all of the features of a conventional Web browser, it does not matter where the Web documents linked to a particular icon are stored; nor does it matter on what type of platform the linked documents are stored. The JWS Browser is able to communicate with the remote platform that hosts the Web document via one of the standard communications protocols supported by the Internet, such as HTTP or FTP. The linked Web documents are automatically downloaded by the JWS Browser (triggered by the JWS program) whenever their corresponding icon is selected. This eliminates many of the complexities in implementing a similar feature linking icons to remote, executable documents.

Each Web document, initially stored locally or remotely before being loaded into memory, includes two elements: a title and a set of references to its components. A Web document can also include embedded files. As with the links, the references in the Web documents can be to remote or local files. In either case, they are handled by the JWS Browser in the same manner as the links. In the User Interface, a reference can be to a Java applet responsible for handling the operations associated with the icon whose related Web document referenced the applet. In that case, upon retrieving the Web document linked to a selected icon, the JWS Browser automatically pulls in and executes the referenced applet, which could have been stored on a remote system. The applet, running in the JWS Browser's virtual machine, can then implement the icon's associated operations without concern about network and operating system complexities, handled respectively by the local operating system and the JWS Browser.

A single JWS applet is referenced in each Web document. This single applet controls or directly implements all of the functions associated with one icon. For example, a spell checker icon can be linked via a Web document to a remote applet that, once downloaded to the local computer and executed by the JWS Browser, spell-checks the

appropriate document(s). Alternatively, a Web document can reference many applets. For example, an icon can be linked to a Web document that references spell-checker and grammar-checker applets so that both are automatically used by the JWS Browser whenever their icon is selected from the toolbar. In addition to applets, a Web document can reference other types of components, including data and image files.

An applet is invoked in response to the selection of a particular icon from the toolbar. Each icon selection event is monitored by the JWS Browser which, following the selection of the icon, retrieves a link from the icon's specification file. The icon is associated with the Web document via a link, and the Web document is automatically loaded by the JWS Browser. The JWS Browser then loads all of the files referenced in the document and executes the

*"Why not have a programming environment for the Internet that users and developers can employ to assemble programs such as Java applets from modules distributed locally and over the Internet, or to download entire program folders from different Internet nodes?"*

referenced executable files, such as applets. Once active, an applet can control a portion of the display or applet window, where it can display results, dialog boxes and icons facilitating user interaction with the applet's functions and capabilities.

The PMS methods are presented to users as options on menus displayed when their associated applet's icon is selected. For example, the portfolio managers methods are displayed as options on a "Portfolio" menu. In conventional GUI fashion, when one of the methods/options is subsequently selected from its parent menu, that option's submenu or page is then displayed by the JWS and enabled for user interaction. Many of the submenus provide a list of portfolios and projects to which the submenus associated method can be applied. The submenus include Create, Import, Choose, Remove, Run and Copy operations.

The methods of the JWS applets for the portfolio and project manager applets have

unique operational characteristics from the vantage point of users working with their respective menus and submenus. With respect to the portfolio manager, PMS provides four methods that allow a user to "Create", "Import", "Choose" and "Remove" portfolios. Each method interacts with a set of portfolio files, and each of these files can be stored on the local or remote systems, representing one portfolio. A generic portfolio file includes a collection of references to its portfolio's constituent project files. As with other file references, a project reference can be to a locally-stored project where the reference is a local file name ("Name") or to a Web document where the reference is a URL.

A portfolio file can contain project file references for its constituent projects, including an "Applet", a "Standalone" program, a Java "Package", an "Image" and a "Remote" applet. As local projects stored in the user's "home" (i.e., local) directory in the memory, they can be accessed by the user and their corresponding project files referenced by path and file name. For example, a reference to an applet project file can be "/home/Applet.prj". The portfolio file also includes a reference (/lib/SemiRemote.prj) to a project file for a read-only project ("SemiRemote") stored in a library directory on machine A and a reference (http://B.com/Internet.prj) to a project file for a read-only project ("Internet") stored on machine B that can only be accessed over the Internet using the JWS Browser.

Each user has a personal portfolio (with a corresponding portfolio file) containing only projects belonging to that user. When the JWS is initially activated, it brings up the personal portfolio as the current, or active, portfolio. The user can choose another portfolio as the current portfolio by using the portfolio manager's "Choose" method/option. To do so, the user selects the desired portfolio's file name or URL from the Portfolio menus and chooses submenu, listing all available portfolios. The user can then view the projects composing the current portfolio by selecting the portfolio manager icon from the JWS toolbar. The applet being executed displays its results and menus on the applet window.

A user of the JWS can create a new portfolio by selecting the portfolio manager's "Create" option and then entering the name of a portfolio. In response, the JWS calls the Portfolio Create method to create the corresponding portfolio file on the local system, display its name in the toolbar and add

the portfolio's name to the Choose and Remove submenus. The newly-created portfolio has no projects, but the user can add projects in the Project Create submenu or import existing projects into the portfolio with the Project Import menu item.

Once the new portfolio has been created, it can be kept private or published on the Internet for access by others. Users can also import existing portfolios that are not currently in their Portfolio Choose submenus. To import such a portfolio, users first select the Import option listed on the Portfolio menu. This triggers the Portfolio Import method to display an import submenu with a name field where users enter the file name or URL of a portfolio, and an import button for users to click when they have completed the entries. In response, the import method adds the portfolio name to the Portfolio Choose and Portfolio Remove submenus. The JWS also changes the current portfolio to the imported portfolio. Once it is on the Portfolio, choose submenu and the imported portfolio can be worked on like any other portfolio.

To remove a portfolio, a user selects it from the Portfolio Remove submenu. In response, the JWS calls the Portfolio Remove method, removing the selected portfolio from the Choose and Remove submenus without deleting the portfolio's corresponding portfolio file. As such, the user can, at any time, import the portfolio using the Portfolio Import option. Each project file included in a portfolio has a corresponding project file describing the project and containing the project's contents. More specifically, each project file contains:

1. The name of the project
2. The project type [Java applet (APPLET), standalone program (STANDALONE), Java class library (PACKAGE), data file (IMAGE), an imported copy of a remote project of one of the four types or a remote applet (REMOTE)]
3. Project administration information, including whether the source code for the project should be distributed to others requesting the project over the Internet and project options
4. The project contents, which can include the actual project contents and/or a set of references to other project files, enabling multiple levels of embedded projects
5. A run page URL (applicable only for applet projects), which is the URL of the HTML file that includes an applet tag for the applet project. This information determines which of the project methods provided by the JWS can be employed by a user on a particular project.

The JWS provides several methods for

working with projects. These methods are presented to users as options on a Project menu. When a method/option is selected, the JWS displays a corresponding submenu in which the user specifies additional details of the operation. The project methods include: Create, Import, Choose, Edit, Remove, Run, Copy and Paste. The methods allow a user to work with existing projects (local or remote) or to create new projects. In either case, projects always exist in the context of a portfolio. When a project is created, it becomes the current project in the current portfolio. A user can create a Java applet project, a standalone program

project, a Java package project, an image project or a remote project. To create any of these projects, the user first "Chooses" the portfolio with which the project is to be associated. It also selects the "Create" option from the Project menu, whereupon the JWS calls the Project Create method. This method displays the Project Create page, where users select the type of project they wish to create (e.g., Users who wish to create an applet, click on an applet button displayed on the submenu). The user then specifies the name of the project to be created and the local directory in memory where the project's corresponding project

# Bristol 1/2 Ad

file is to be stored. Once the user has specified the attributes for the project, the Project Create methods adds a reference to the project's corresponding project file to the portfolio file of the current portfolio.

When the project being created is an applet, standalone program or Java package, the user may also have access to source code for the newly-created project. The user enters the file names of the corresponding source files on the Project Create page. The JWS adds these source file names to a "Sources" list maintained in memory so the source files can be accessed by the user (e.g., for editing and compilation). The user also enters the name of the main file for the program (i.e., the file containing the "main" routine) of which the newly created project is a part. When the project being created is a Java applet, it is possible that the Java applet is referenced in an HTML page so that when the applet's reference is selected, the applet is executed. The JWS allows for such relationships represented via a Run Page URL field in the Project Create page where the user optionally enters the name of the HTML page executing the applet.

When creating an image project by Choosing "image" from the Project Create submenu, the user enters the name of the image project and the URL of the corresponding image file. The user can then optionally enter attributes associated with the image, such as:

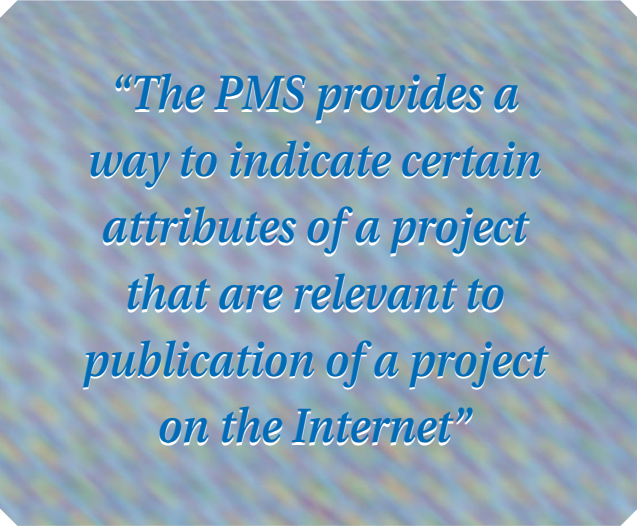
1. The image's alignment with respect to surrounding text (e.g., choosing "bottom" alignment causes a browser displaying the image to align the bottom of the image with the bottom of the text)
2. Whether the image is active; where a person viewing the image can click on different regions of the image to produce different actions
3. An optional text string to be displayed in lieu of the image by browsers not able to display the image

Once this information is provided, the user clicks on the "Apply" field of the Project Create page. The Project Create method makes the newly created image project the current project and displays the image in the Applet window. Additionally, the Create method adds the image project name to the Choose, Edit and Remove submenus in the Project menu. It also adds the name of the corresponding project file to the portfolio file associated with the current portfolio.

Users can import any type of project into one of their personal portfolios by choosing a portfolio as the current portfolio,

selecting the "Import" option from the Project menu and entering the name or URL of the project to be imported in the Project Import page displayed by the Project Import method. After entering the necessary information, the user clicks an "import" button displayed on the import page. The import method imports the designated project into the current portfolio and adds the project name/URL to the Project Choose, Edit, Remove and Run submenus. The import method also adds the name of the imported project's project file to the current portfolio if it is not already contained therein. The JWS does not make the imported project the current project, but displays the imported project if the user subsequently selects the Portfolio Manager icon from the toolbar.

The JWS allows a user to create a remote applet project by "Choosing" the current portfolio, selecting the "Create" option from the "Project" menu and clicking the



*"The PMS provides a way to indicate certain attributes of a project that are relevant to publication of a project on the Internet"*

"remote applet" button displayed on the Project Create submenu. The user enters the name of the project and the URL of the HTML page executing the applet. Then the user exits the Project Create submenu by clicking on "Apply". After that, the create method creates a project file with empty contents and a run page URL field set to the URL of the HTML page executing the applet. For example, the remote project file has a run page URL set to the URL ("http://C.com/RunApplet2.htm") of the Web page ("RunApplet2.htm") running the remote applet "Applet2". The create method also adds the name of the project file to the current portfolio's portfolio file. The JWS then makes the imported project the current project, loads the Portfolio Manager and selects the current project to be displayed by the Portfolio Manager. After that, the JWS adds the imported applet project's name to the Choose, Edit, Remove, Run and Copy submenus in the

Project menu.

A user can then run the remote applet by selecting its name from the Project Run submenu or by loading the Portfolio Manager, selecting the remote project and then pressing the Run button on the toolbar. After that, the Project Run method passes the URL of the Web page, referenced in the run page URL field of the remote applet project file, to the Web browser for downloading the referenced Web page (<http://C.com/RunApplet2.htm>) and running the remote applet (Applet2).

If a Run page URL in an applet's project file is not specified, that applet project can still run using the Project Run method. The Project Run method automatically generates a new Web page containing an applet tag created with the project attributes and parameters entered by the user on the Edit Project run folder. This automatically generated HTML page is loaded into the JWS, which uses the browser to run the applet project. This feature allows users to execute applets without having to know the HTML syntax for referencing applets.

The Project Manager's Copy method allows a user to copy an applet into an HTML file, executing the applet without requiring the user to know the HTML syntax for referencing applets. The user first selects an applet project in the current portfolio and selects the Copy option from the Project menu. The Copy method then copies the contents of the selected applet project to a clipboard maintained by the JWS. After that, the user selects the Text Editor Icon from the toolbar, whereby the JWS executes the Editor method. The Editor method displays a text editor containing an Edit menu, including a list of editing options such as "Paste". The user selects the "Paste" option from the Edit menu, upon which the paste method pastes the contents of the clipboard (i.e., the applet being copied) into a new file. The user can then save the new file as an HTML file, causing the JWS to add the appropriate links of the copied applet to the saved HTML file. As with other new projects, the JWS adds the file name of the new HTML file to the current portfolio's portfolio file. Alternatively, users can simply drag the image of the applet to be copied onto the image of an HTML file that they wish to include the applet. The JWS will then copy the contents of the applet to the designated HTML file and add tags referencing the copied applet to the HTML file.

The Project Edit method also allows a user to edit projects of all types. The Edit method can be invoked by a user of the JWS

in one of two ways. First, the user can click on the Edit Project icon IA3 displayed on the toolbar to invoke editing (i.e., the editing method) on the current project. Secondly, the user can select the name of the project to be edited from the Project Edit submenu. Once editing is selected for a designated project, the JWS Editor method opens an edit page that includes six folders in which the user can edit information for the designated project. These six folders and their associated information include:

1. General information about the project, including name, type and source directory
2. Build information needed to compile the project
3. Debug/Browse information needed to debug and browse source files
4. Run information for executing an applet or standalone program in the JWS Browser
5. Publish information to allow the project to be copied by other users
6. Portfolio information needed to display the project in the Portfolio Manager

The Project Edit method allows a user to edit fields in these six folders only where appropriate. To assist the user, the edit method grays out inapplicable fields depending on the type of the project being edited and whether the project is local or remote. The JWS allows users to provide information for remote as well as local projects identified by file names or URLs to specify, for example, that the source code for a particular project to be debugged or browsed exists on some remote node.

The JWS allows users to employ portfolios and projects from remote sources and to publish their own portfolios and projects for others' use. Thus, the PMS provides a way to indicate certain attributes of a project that are relevant to publication of a project on the Internet. These publication attributes are contained in the Publish and Portfolio Folders. The Portfolio Folder includes the following fields:

- **Description:** A brief description of the project displayed by the JWS Browser when the mouse is positioned over the project image in the Portfolio Manager
- **Portfolio image:** The URL for the image file (GIF, JPEG or other) that URL represents the project image in the portfolio
- **Features:** The general characteristics of the project, like whether the project is video, graphics or audio.

The Publish Folder includes the following fields:

- **Distribute source:** A toggle field that controls copies where the project's source

files are copied when the project is copied from one portfolio to another

- **Submitter Name:** The name, e-mail address and Web page of the person
- **E-mail and URL:** Adding the project to the portfolio

Users can change the current project in one of two ways. In the first, users start by selecting the Portfolio Manager icon from the JWS toolbar. This causes the JWS to open a Portfolio display which shows the projects of the current portfolio in the Applet Window. Users then select the project they want to be the current project from the Portfolio display. The JWS makes the selected project the current project and displays the name of the current project on the JWS toolbar. Alternatively, the user can change the current project by choosing the project name from the Project Choose submenu.

The JWS allows a user to remove a project from a portfolio in one of the following ways. First, in the portfolio manager display in the applet window, users can select the project they wish to remove and then click a Remove Icon in the Portfolio Manager. Alternatively, users can choose the name of the project to be removed from the Project Remove submenu. In either case, once the user has indicated the project to be removed, the Project Remove method removes the project from the Choose, Edit, Remove and Copy submenus of the Project Menu. The Project Remove method does not delete the removed project's project file to ensure that the user can subsequently import the project at a later time if required.

The system detailed above is one implementation of the PMS system as described in the '502 patent. Other implementations of the PMS system are possible and may be governed by the scope of the claims of the '502 patent. As such, developers or companies who believe they have the same or similar systems should carefully review the '502 patent and seek competent legal advice to ensure their systems do not infringe the '502 patent.

**NOTE:** This article is not intended as legal advice and should not be used as such. ☛

#### About the Author

Michael Zarrabian is a registered patent attorney who specializes in Protecting Creative Intellectual Property<sup>SM</sup> through Patent, Trademark, Copyright, Trade Secret, Licensing, Internet and Computer laws. He has a BS and MS in Computer Engineering with eight years of hardware/software engineering experience. Michael can be reached via e-mail at michaelzz@worldnet.att.net



michaelzz@worldnet.att.net

**SYS-CON PUBLICATIONS**

**PHONE, ADDRESS & WEB DIRECTORY**

**CALL FOR SUBSCRIPTIONS  
1 800 513-7111**

International Subscriptions  
& Customer Service Inquiries  
**914 735-1900**

or by fax: 914 735-3922

E-Mail: [Subscribe@SYS-CON.com](mailto:Subscribe@SYS-CON.com)  
<http://www.SYS-CON.com>

MAIL All Subscription Orders or  
Customer Service Inquiries to

**JAVA DEVELOPER'S JOURNAL**

Java Developer's Journal  
<http://www.JavaDevelopersJournal.com>

**VRML DEVELOPER'S JOURNAL**

VRML Developer's Journal  
[VRMLJournal.com](http://VRMLJournal.com)

**NLC National JAVA LEARNING CENTER**  
National Java Learning Center, Inc.

**JAVA DEVELOPER'S JOURNAL**  
1997 JAVA Products & Services  
**Buyer's Guide**  
& Internet Directory  
JDJ Buyer's Guide  
[JavaBuyersGuide.com](http://JavaBuyersGuide.com)

**WEB-PRO DEVELOPER'S SUPPLEMENT**

**Web-Pro Developer's Supplement**

SYS-CON Publications, Inc.  
39 E. Central Ave.  
Pearl River, NY 10965 – USA

**EDITORIAL OFFICES**  
Phone: 914 735-1900  
Fax: 914 735-3922

**ADVERTISING & SALES OFFICE**  
Phone: 914 735-0300  
Fax: 914 735-7302

**CUSTOMER SERVICE**  
Phone: 914 735-1900  
Fax: 914 735-3922

**DESIGN & PRODUCTION**  
Phone: 914 735-7300  
Fax: 914 735-6547

**DISTRIBUTED in the USA by  
International Periodical Distributors**  
674 Via De La Valle, Suite: 204  
Solana Beach, CA 92075  
Phone: 619 481-5928

**Worldwide Distribution by  
Curtis Circulation Company**  
739 River Road,  
New Milford NJ 07646-3048  
Phone: 201 634-7400

## Rogue Wave Software Reports Fiscal Results

(Boulder, CO) – Rogue Wave Software has reported financial results for its second fiscal quarter. Revenue for this quarter was \$10.3 million, up 34% from the \$7.7 million achieved in the second quarter of 1997. Excluding the \$1.2 million non-recurring Stingray Software acquisition and headquarter relocation expenses, net earnings for the quarter were \$479,000 or \$0.04 per share. After non-recurring expenses, the company experienced a net loss of \$307 thousand, or \$0.03 per share.

“The key event of this quarter was the acquisition of

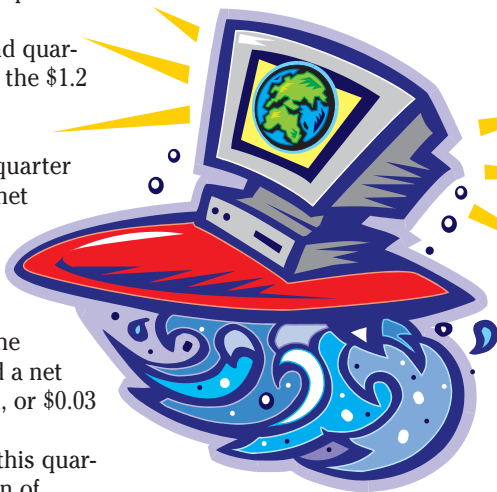
Stingray Software, Inc.,” said Thomas Keffer, Rogue Wave’s CEO and Chairman of the Board. “Now that we have had

a chance to look more closely at the human and product resources that Stingray has brought to Rogue Wave, we are more excited than ever by this merger. We are looking forward with great excitement to working with this outstanding

team.” Keffer added: “Rogue Wave and Stingray Software have already announced their first integrated product,

StudioJ, a suite of Java components.”

The company’s investor relations department can be reached by e-mail at [ir@rogue-wave.com](mailto:ir@rogue-wave.com), by Fax at 303 443-7780 or via the company Web site at [www.rwav.com](http://www.rwav.com).



### IBM Tuning JavaOS™ for Business™ to Network Computers

(Austin, TX) – IBM has announced that it is tuning the JavaOS™ for Business™ product for Network Computers (NCs) based on Intel processors with assistance from Intel. IBM and Intel are also working together so that the JavaOS for Business product supports Intel’s Lean Client Guidelines. By bringing together the Intel Architecture with the JavaOS for Business software, IBM and Intel are cooperating to expand the choices available to customers seeking network computing solutions.

In addition, IBM will develop a version of the IBM Network Station network computer utilizing an Intel microprocessor for the high end of the Network Station family. The Intel-based NC from IBM is expected to be available in the first half of 1999 and will take advantage of the JavaOS for Business product. IBM’s Network Station will be compliant with both the Intel Architecture Lean Client

Guideline and the Network Computer Reference Platform.

To get information about any IBM software, go to [www.software.ibm.com](http://www.software.ibm.com).

### Those in the Know, Know Apptivity!

(Burlington, MA) – Java™ is quickly emerging as the new language of choice for building and deploying business applications. Progress Software’s Apptivity, a Java application development environment, is putting Java to work for business applications, providing the fastest way to build, deploy and maintain mission-critical database applications across a mix of platforms.

Industry analysts at the Gartner Group, Aberdeen Group, DataPro, IDC and others have complimented Progress Software on the innovative nature and value of its Apptivity product.

For more information, please contact Margot Delogne by phone at 781 280-4000 or e-mail her at [delogne@bedford.progress.com](mailto:delogne@bedford.progress.com)

### Novera™ Introduces New Java™ Application Server

(Burlington, MA) – Novera Software, Inc. has announced the most complete Java application server as the foundation for the company’s new jBusiness™ Solutions. Novera revealed the Novera Application Server at the JavaOne show in San Francisco.

jBusiness Solutions are

about Java-enabling the core business process to improve the quality of development, reduce cycle time and get more results from limited resources. jBusiness brings together the necessary products, services, consulting, training, partners and Java experts to help companies easily and successfully develop, deploy and manage server-side Java applications.

For more information, contact Christina Pandapas at 978 474-1900 or via e-mail at [cpandapas@pancomm.com](mailto:cpandapas@pancomm.com).

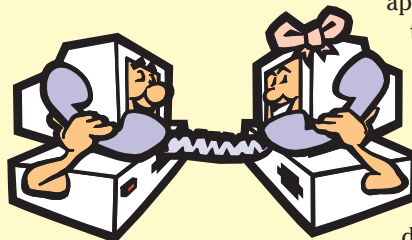
### Digitivity™ Announces Server-Centric Software

(San Francisco, CA) – Digitivity, Inc., developer of the Applet Management System, has announced the expanded capability of its policy-based management tool to manage and securely deploy full Java applications. The expanded capabilities of this policy-based tool give network administrators the ability to manage all mobile code in an enterprise network, including ActiveX, JavaScript and plug-ins.

For more information, contact Andi Bruno by phone at 650 947-1904 or via e-mail at [andi.bruno@digitivity.com](mailto:andi.bruno@digitivity.com). Digitivity’s Web site is at [www.digitivity.com](http://www.digitivity.com).

### Volano Releases Open Chat Platform

(San Francisco, CA) – Volano LLC has released extensions to VolanoChat™ and VolanoChatPro™, the company’s industry leading 100% Pure Java™ chat platforms, which allow Web developers to easily create interactive business applications. The chat platforms include open interfaces so that any Java



applet can be embedded in the chat client applet and any Java servlet can be accessed through the chat server.

Volano has also established a partner program for Java developers. Volano mar-

kets third party applets and servlets to the company’s international customer base. Java developers should contact Volano for more information on distribution, OEM and reseller opportunities. Call 415 587-4297 or visit Volano’s Web site at [www.volano.com](http://www.volano.com).

# Sales Vision

## Intertop Announces Internet-Development Environment Solution

(Saratoga, CA) – Intertop Corporation has announced its flagship product, i-Xpresso. Engineered specifically for the Internet, businesses and developers can get the environment they've required for Internet applications. Browser- and Java-version independent, Intertop's proprietary technology, i-Enable, includes a small footprint, Virtual Machine Extender (i-VMX) that extends the already popular reach of Java, allowing it to run without software residing on the client side. The Internet-application development technology within i-Xpresso enables multi-tier interactive communication on the Net.

For more information, visit Intertop's Web site at [www.intertop.com](http://www.intertop.com) or call toll free at 888 873-2420. ☛

## Zero G Ships InstallAnywhere 2 Express

(San Francisco, CA) – Zero G Software has introduced InstallAnywhere 2 Express, a basic version of the industry's most popular installer for Java software. InstallAnywhere 2 Express incorporates the essentials of the feature set found in

## Jumping Beans™ Brings True Applications Mobility to Enterprise Environments

(San Francisco, CA) – Ad Astra Engineering has announced the arrival of true application mobility via their Jumping Beans™ technology.

Jumping Beans, based on a concept of Robert Orfali and Dan Harkey, and described in their book *Client/Server Programming with Java™ and CORBA*,



is a software framework that allows a developer to mobilize a Java application. The mobilized Java application can then automatically move from one host to another during its lifetime. As the application moves from host to host, it carries with it all of the essential information so it

is able to execute on hosts that did not have the application previously installed.

To inquire about Jumping Beans and the Jumping Beans beta program, contact Chris Rygaard at Ad Astra Engineering, Inc. at 408 738-4616 or e-mail him at [crygaard@AdAstraEng.com](mailto:crygaard@AdAstraEng.com). Check out the Jumping Beans Web site at [www.JumpingBeans.com](http://www.JumpingBeans.com). ☛

the InstallAnywhere 2 Standard Edition but at a fraction of the cost. This new version is ideal for small, independent yet cash-conscious Java developers who want to quickly and easily deploy custom software to a limited number of target users.

Like the Standard Edition, InstallAnywhere 2 Express allows developers to shave days or weeks off the overall application development process. Written entirely in Java, Express makes Java applications double-clickable and performs automatic Java virtual machine installations without running a separate installer. For more information, contact Carrie Smith at 415 512-7771, ext. 24 or via e-mail at [carrie@zerog.com](mailto:carrie@zerog.com). ☛

## Specialized Software International Announces the Next Wave in JavaBean™ Technology

(Boston, MA) – Specialized Software International has announced the release of ROAD:BeanBox, a rich set of user interface (UI) and data access components that adhere to the JavaBeans™ API. These components enable the access and data manipulation of JDBC-compliant database servers with virtually no programming. ROAD:BeanBox offers a data-cursor engine that provides developers in any IDE with lightweight, cross-platform data access and data-caching technology.



For additional information on Java development using ROAD:BeanBox, or for purchasing information, visit the ROAD:BeanBox Web site at [www.BeanBox.com](http://www.BeanBox.com). For additional information on Specialized Software International and its products and services, visit the Specialized Software Web site at [www.SpecializedSoftware.com](http://www.SpecializedSoftware.com). ☛

## KL Group Releases Version 3.0 of JClass Products

(Toronto, CANADA) – KL Group Inc. has announced that its entire family of JClass products has been upgraded to version 3.0, featuring compatibility with

the JavaSoft "Swing" component toolkit in the JDK 1.1 and the forthcoming JDK 1.2. The 3.0 version offers a synchronizing release of JClass BWT, Chart, Field and LiveTable, ensuring that developers can easily identify compatibility with the latest JDK.

JClass 3.0 continues the tradition of providing common API support across multiple JDK environments. Developers using JClass in JDK 1.0.2 will find it easy to migrate their applications to JDK 1.1. Developers beginning to work with the newly released JFC 1.1/Swing component kit and the JDK 1.2 beta are now supported by the latest JClass releases.

For additional information, contact Lee Garrison via e-mail at [lee@klg.com](mailto:lee@klg.com) or by phone at 800 663-4723, ext. 769. ☛

## LPC Delivers Persistence Framework for Java™

(Toronto, CANADA) – LPC Consulting Services has announced the production release of its JdbcStore product, a persistence framework for Java. From the same developers who created ODBTalk, the leading Smalltalk database framework for ODBC, JdbcStore provides the Java programmer with state-of-the-art OO persistence for complex client/server and Internet/intranet applications.

For more information, call Ken Findlay at 416 787-5290, e-mail him at [kfindlay@ilap.com](mailto:kfindlay@ilap.com) or check out LPC's Web site at [www.ilap.com](http://www.ilap.com). ☛

## The Object People Announces the Release of TOPLink™ for Java™

(Ottawa, CANADA) – TOPLink™ for Java™ is an industrial strength, object-to-relational mapping tool that has just recently been announced for release by The Object People.

TOPLink technology has been in use since 1991. It bridges the gap between the world of objects and the world of relational technology by mapping objects to relational tables. TOPLink incorporates features that are normally found only in

object databases, such as object-level transactions and queries. This simplifies the application development process because it allows developers to work exclusively with objects. No SQL programming is required.

For more information on TOPLink, visit The Object People Web site at [www.objectpeople.com](http://www.objectpeople.com), or else call Bill Allen at 919 852-2200 or e-mail him at [ballen@objectpeople.com](mailto:ballen@objectpeople.com). ☛







# Java Skeptics Run Amuck

## THE GRIND

by Java George

***“critics who can’t tell the difference between business strategy and computer programming”***

*Java George is George Kassabgi, director of developer relations for Progress Software’s Appivity Product Unit.*

Some of the recent attacks on Java are so misinformed that, at first, I thought they were intended as a parody of the Java tirades coming out of the Microsoft PR office. Then I realized they were meant to be serious.

Take the complaint that Java doesn’t work because the Corel personal productivity suite failed. We might as well complain that the internal combustion engine is worthless because Ford’s Edsel was a bust.

Corel is a very nice company and their graphics products are world class. But, in an attempt to break out of the graphics tools niche and get into the much broader personal-productivity market, the company went out and bought a very old suite. Corel’s impulse to expand its market was probably right, but its choice of the vehicle to do it was misguided from the start. The company found itself saddled with old code. So, it tried to rebuild the product from the ground up using Java, and it didn’t work.

From the start, the Corel effort was a very risky endeavor for many reasons, the least of which was Java. The critics who see Corel’s failure as a failure of Java itself must not only be smoking something, but inhaling. Should we consider the failure of the initial Microsoft Network, a distant also-ran in the online access market, as reason to avoid all the Microsoft technologies that went into it? We all better hope not. Similarly, Oracle killed its Project Sedona. Does that sound like the death knell for BASIC, C and C++? Don’t bet on it.

But, we can learn some things from Corel’s experience with Java and its personal productivity suite. First, a personal productivity suite is client-only software (and a ton of software — millions of lines — at that). While Java can work quite well for client-only software, it is really designed for building fully distributed applications. Corel turned to Java because it would give it a way to differentiate the product in a saturated market already hopelessly dominated by a monopoly product. The project, however, never took advantage of Java’s strengths, such as application partitioning, easy network-ability and easy distribution of updates.

Second, the Corel experience confirms what most of those who work regularly

with Java already know — that the technology still has some maturing to do, and Java development works best in the hands of experienced Java programmers. Java is easier to program in than, say, C++, but that doesn’t mean you should give it to just anybody and expect them to knock out a few million lines of effective code. The right tools are needed to build successful business applications with Java. VI and EMACS alone simply don’t cut it.

Then there are the Java pundits who proclaim its death because Netscape is seemingly retreating from Java. Again, we are dealing with critics who can’t tell the difference between business strategy and computer programming.

Netscape’s problems have more to do with trying to make a buck when your competitor is giving the product away for free. The company was trying to move ahead on too many fronts at once — IFC, a worthwhile foundation class for the Java UI; the Kiva application server platform; and Communicator — all the while battling Microsoft in the browser market. Netscape was forced to pull back on its Java efforts because of its financial situation, not because Java wasn’t working for it or its customers.

The upshot: For now there are two primary Java versions, the actual Javasoft/Sun version and whatever proprietary version Microsoft settles on. Over time there will be more Java VMs designed specifically for platforms such as IBM’s AS/400, HP-UX, Solaris, WindowsCE, JavaRing and more, which will only work to the benefit of the Java platform. The skeptics don’t quite understand this. The skeptics also were sure that multiple versions of SQL would ruin that standard. Yet the multiple versions have strengthened it. The same will happen with Java.

As for Netscape, life is tough for small companies, especially when they are battling an industry giant. Although Netscape has slowed its Java R&D, don’t count Netscape out. In the meantime, you can expect Java technical advances to continue to pour out of Sun, IBM and others, regardless of what Netscape or Microsoft does.

Now that we’ve reined in the Java skeptics, the next Java George column will look into some of the legitimate issues surrounding Java. ☛



Joe@sys-con.com

Ad

# KL Group Full Page Ad