JAVA™ DEVELOPER'S JOURNAL

The World's Leading Java Resource

August 2001 Volume: 6 Issue: 8

JAVADEVELOPERSJOURNAL.COM

SYS-CON MEDIA

JAVA DEVELOPER'S JOURNAL

INTERNATIONAL ADVISORY BOARD

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

EDITORIAL

EDITOR-IN-CHIEF: ALAN WILLIAMSON
EDITORIAL DIRECTOR: JEREMY GEELAN
ASSOCIATE ART DIRECTOR: LOUIS F. CUFFARI
J2EE EDITOR: AJIT SAGAR
J2ME EDITOR: JASON BRIGGS
PRODUCT REVIEW EDITOR: JIM MILBERY
FOUNDING EDITOR: SEAN RHODY

PRODUCTION

VICE PRESIDENT, PRODUCTION AND DESIGN: JIM MORGAN
EXECUTIVE EDITOR: M'LOU PINKHAM
MANAGING EDITOR: CHERYL VAN SISE
EDITOR: NANCY VALENTINE
ASSOCIATE EDITORS: JAMIE MATUSOW
GAIL SCHULTZ
BRENDA GREENE
ASSISTANT EDITOR: LIN GOETZ
TECHNICAL EDITOR: BAHADIR KARUV, PH.D.

WRITERS IN THIS ISSUE

SCOTT AMBLER, BILL BALOGLU, JACQUIE BARKER, JASON BRIGGS, JOHN CRUPI, JEREMY GEELAN, JUSTIN HILL, TONY LOTON, ROB MACAULAY, SANJAY MAHAPATRA, KIMBERLY MARTIN, BILLY PALMIERI, STEVEN RANDOLPH, AJIT SAGAR, SANJAY SARATHY, ANTHONY SIMMONS, ALAN WILLIAMSON, BLAIR WYMAN

SUBSCRIPTIONS:

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

SUBSCRIPTION HOTLINE: SUBSCRIBE@SYS-CON.COM

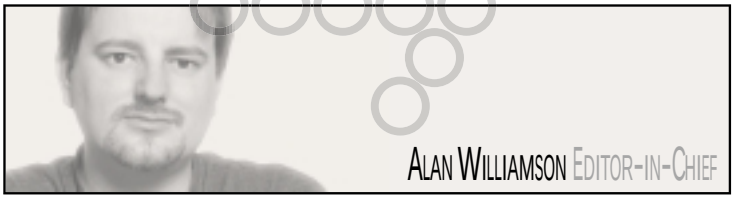
COVER PRICE: $4.99/ISSUE
DOMESTIC: $69.99/YR. (12 ISSUES)
CANADA/MEXICO: $79.99/YR. OVERSEAS: $99.99/YR.
(U.S. BANKS OR MONEY ORDERS). BACK ISSUES: $10/EA., INTERNATIONAL $15/EA.

EDITORIAL OFFICES:

SYS-CON MEDIA 135 CHESTNUT RIDGE RD., MONTVALE, NJ 07645
TELEPHONE: 201 802-3000 FAX: 201 782-9600
JAVA DEVELOPER'S JOURNAL (ISSN#1087-6944) is published monthly (12 times a year) for $69.99 by SYS-CON Publications, Inc., 135 Chestnut Ridge Road, Montvale, NJ 07645. Periodicals postage rates are paid at Montvale, NJ 07645 and additional mailing offices. POSTMASTER: Send address changes to: JAVA DEVELOPER'S JOURNAL, SYS-CON Publications, Inc., 135 Chestnut Ridge Road, Montvale, NJ 07645.

WORLDWIDE DISTRIBUTION BY:

CURTIS CIRCULATION COMPANY
NEW MILFORD NJ



ALAN WILLIAMSON EDITOR-IN-CHIEF

We All Need a Week Off in the Sun

At last we can return to sanity. The speeches are over, the bunting is down, and the mad hysteria is at an end. After the chaos of JavaOne we can return to normal. I am of course paraphrasing that great literary character Edmund Blackadder. It has now been a month since I arrived home from our annual pilgrimage to JavaOne and I'm still following up on all the business cards that were thrust in my general direction.

JavaOne is a great opportunity to do some serious catching up and general networking, and it was a real treat as Blair Wyman and I took some time out to have a chat with James Gosling about what he's been up to lately. You can read what we spoke of elsewhere in this issue. It was interesting to talk with James concerning the overall direction of Java and discover how he feels about many of the issues. We touched on the old debate regarding Java being open-sourced and his response was so wonderfully practical that I can't help feeling that maybe he resents all the bickering and politics that divert attention from the underlying beauty that makes Java, well…Java.

The industry is still struggling to find its feet after the dot-bomb exploded, and desperately trying to stave off redundancies is the very company behind Java: Sun Microsystems. Although we probably shouldn't read too much into this (as Scott McNealy keeps telling us), the core business at Sun is still manufacturing and selling hardware, and the work Sun does with Java is still very much full steam ahead. That said, as you'll read from our chat, James does comment on the fact that "all the cool stuff with Java isn't done at Sun," so even if they are hitting a rocky patch in the road, Java isn't going to suffer much.

This was the month that Sun had their famous one-week nonpaid holiday – a move to save some pennies, which for an organization of Sun's size and payroll would account for quite a lot of pennies. But you can't help wondering how many of the engineers just went into work anyway.

Looking up the coast from Sun to another company's trials and tribulations, this month we saw that Microsoft was saved from the chopping block and has been allowed to stay together as a single operating unit. I can't really comment on whether this is going to be a good thing or a bad thing, but I'm getting a little paranoid regarding Microsoft's influence on the desktop market now that Windows XP is just around the corner. For example, have you been following the SmartTag debate? This is where Microsoft will put hyperlinks around special keywords it finds in all rendered HTML pages to link you to another site with more information.

For example, say your Web page happened to mention Sun Microsystems somewhere in the text. A small link made out of the text would – if you were to position your mouse over a pop-up – display more information and optionally take you to another site…all without the approval or knowledge of the original Web site producer. In principle it's a great idea, but one that is open to so much abuse. The upshot is that Microsoft has postponed this feature's appearance in IE for a little while longer, but rest assured it'll be there in some version, at some point.

That aside, what annoys me most about Microsoft's newest addition to its operating system suite is that there's still no embedded Java Virtual Machine. With Microsoft's strong relationships with the major PC manufacturers, Windows XP will find its way into the homes of millions of users who will simply never think of installing a JVM, let alone replacing the operating system. Due to the high-level politics and strategies of two companies, millions of users are affected, including a whole development community that won't be able to service this new user base.

We need to start moving Java applications into the mainstream and have sites chockful of executable JAR files that users simply 6and run without worrying about the requirements they face in the README.TXT file beforehand.

Maybe if we had this, Microsoft wouldn't seem half so bad.

alan@sys-con.com

AUTHOR BIO

Alan Williamson is editor-in-chief of Java Developer's Journal. *In his spare time he holds the post of chief technical officer at n-ary (consulting) Ltd, one of the first companies in the UK to specialize in Java at the server side. Reach him at alan@n-ary.com (www.n-ary.com) and rumor has it he welcomes all suggestions and comments!*

WRITTEN BY SANJAY SARATHY

The Ring Is Bloody

With too many solutions looking for a business problem to solve

The Thrilla in Manila: both the name and the events of that steamy October day in 1975 remain seared in the memory of all who watched it. Muhammad Ali and Joe Frazier, two of the greatest boxers in heavyweight history, battled toe-to-toe for 14 rounds, until Frazier's corner surrendered and threw in the towel.

Today the technology industry is in the midst of its own thrilling fight – one that is centered on the much-hyped world of Web Services. And just as with Ali and Frazier, the various combatants are treating each other to verbal jabs and taunts. Anyone who attended JavaOne this past June witnessed this, particularly at some of the morning keynotes. Unfortunately, as companies continue to spar over the issue, we are losing the necessary focus on the business impact of Web Services as each vendor beats its own technology drum.

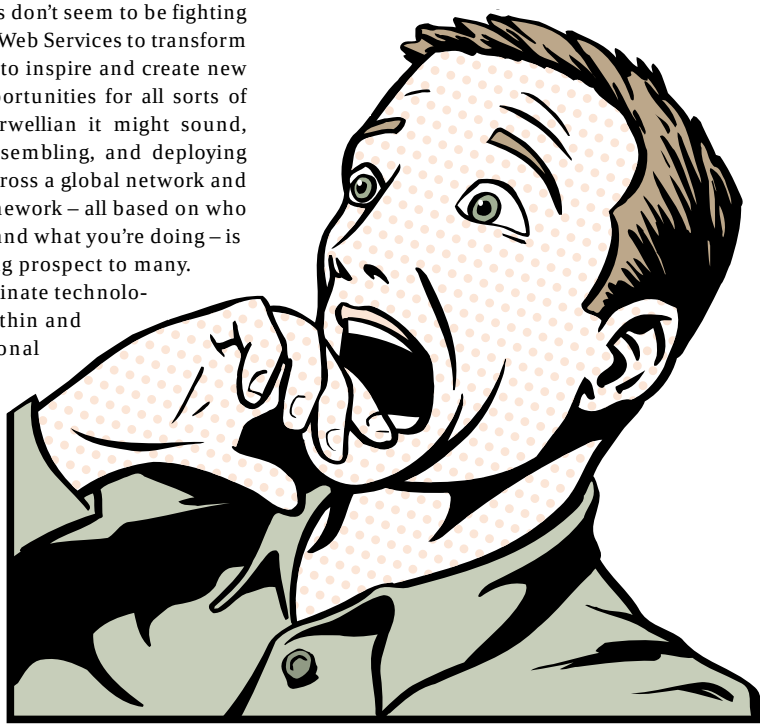
One thing companies don't seem to be fighting about is the potential of Web Services to transform how technology is used to inspire and create new revenue-generating opportunities for all sorts of businesses. However Orwellian it might sound, the idea of creating, assembling, and deploying personalized services across a global network and within a contextual framework – all based on who you are, where you are, and what you're doing – is an enormously appealing prospect to many.

The potential to eliminate technology and business silos within and across organizational boundaries provides another compelling argument for Web Services, especially when companies realize they won't have to abandon the various investments they've already made.

Companies agree on something else as well: no matter what the technology flavor of the month might be, all businesses want to create tighter synergies throughout their entire value chain that encompass customers, employees, suppliers, and partners.

This is truer today than ever. These synergies have taken on a whole new importance thanks to the explosion in online personalization; a development that now touches individual consumers as well as larger business-to-business operations.

As vendors, we often get so distracted by the technology details that we end up losing sight of this bigger picture. Worse yet, perhaps, this too often makes us adversaries, leading to wasted time and energy as we throw punches at each other about which is the best way to proceed. We have to remind ourselves that organizations create real value when they're able to implement



sarathy@iplanet.com

AUTHOR BIO

Sanjay Sarathy is director of product marketing at iPlanet. He holds a BA in quantitative economics from Stanford University and an MBA from the Haas School of Business at the University of California.

I'm a subscriber to ***JDJ*** and am very happy with the magazine (I'm from the U.K. living and working in Germany). The digital edition is an excellent bonus for me as the post takes awhile. The content of the magazine is simply first class, both in quality and choice of subject matter – I'll definitely renew my subscription when the time comes!

Another publication that interests me is ***WBT***, which I currently read online only. I'd be interested in receiving subscription information/pricing for oversea subscriptions for this publication (possibly just the digital edition).

In the meantime, keep up the excellent work, and I look forward to reading the next issue(s)

Ian Harvey
ian.harvey@tui.de

Plaudits for Tony Loton

I've been a subscriber of ***Java Developer's Journal*** for the past two years. I read your applet-servlet communication article ["Fitting the Pieces into the Enterprise Java Jigsaw", ***JDJ***, Vol. 6, issue 5] and I have a similar kind of job requirement. I'm following your example in the article of passing a bean from the servlet to the applet by trying to pass a servlet session object to the applet, but it's not working. However, when I tried to pass my bean from the servlet to the applet, it worked.

I know the servlet runs at the server side and also the session, but the applet runs at the client. Can we pass the servlet-session object to the applet ?

I've been struggling with this problem for the past month and my friends couldn't help me. I would appreciate it if you could throw some light on the subject.

Thanks a lot.

Mohan Palaniswamy
mpalaniswamy@ccsin.com

Hi Mohan,

I hope the original article was useful to you. To answer your question, an object may be passed from the server to the client only if it's serializable, i.e., implements the java.io.Serializable interface. Any object not implementing Serializable cannot be passed.

Serialized objects are passed "by value" across the network so that the receiver sees a duplicate version of the original rather than the original. So even if you could pass the HttpSession, it would no longer reflect the current session state as kept current on the server. Presumably you want the client to see some of the information held within the session. How about defining your own object to hold values extracted from the session, make it Serializable, and exchange that object instead?

Incidentally, I often see this problem when someone is trying to pass a JDBC connection from server to client. It would be nice if you could, but I'm afraid not.

Tony Loton
tony@lotontech.com

The ***JDJ*** that you gave me at the conference was quite impressive and very informative. Up until that time I was a regular reader of *Java World*, but after going through ***JDJ*** I think I might change the magazine for my Java requirements.... :)

Ashutosh Bhonsle
bhonsle@hotmail.com

I was pleased to see the JNI article in the June 2001 ***Java Developer's Journal*** since it dealt with the IBM iSeries (AS/400) platform. As a Java developer using the iSeries it's nice to see it get some press. It seems to be a well-kept secret although it's a very robust platform for running Java applications.

I hope to see more articles dealing with the IBM iSeries platform.

T.Allen
tballen@copart.com

The Ring Is Bloody

<INSERT TECHNOLOGY CHOICE OF THE MOMENT> and profitably meet their business objectives.

This is even more important today given the current economic environment. Technology vendors must be able to articulate how they enable their customers to generate maximum return on their critical assets, which include their people, their business processes, and their information systems.

During and since JavaOne, I've had the opportunity to speak with a number of companies developing interesting technologies devoted to various aspects of the Web Service paradigm. I don't doubt that some of these ideas will be revolutionary. I'm confident as well that a few of the people involved in these developments will become wealthy, provided, of course, their companies get funded. Still, I could not help but feel overwhelmed by just how many different types of solutions are looking for a business problem to solve – even for someone like myself, who is living and breathing this stuff!

I pity the poor CIO, line-of-business executive, or even developer, who, having just figured out how to really take full advantage of the Java 2 Platform, Enterprise Edition (J2EE) model, must now wrestle with this dazzling array of new technologies.

In the end, a vendor's credibility is only partly the result of an innovative technology getting shipped to market. After all, who can't name a handful of cool companies that got knocked out before ever delivering a product? No, a vendor's credibility is more a product of their ability to help businesses understand how to generate top - and bottomline – benefits from the technology they've chosen. What's more, this model is true whether the technology trend was yesteryear's client server model (may it rest in peace) or today's promising Web Services.

AJIT SAGAR J2EE EDITOR

J2EE Design Patterns: The Next Frontier

We live in a world where abstraction is the name of the game. I used to be an avid reader of Asterix comics, and thinking of abstraction reminds me of a couple of panels in the "Obelix & Co." comic book. A Roman emissary tries to explain to the (simpleton) Obelix the intricacies of Roman economy in simple words: "Make much menhir, get much gold," or something to that effect.

I don't mean to say that we developers are simpletons. But, as my colleagues in marketing say, the presentation needs to be "dumbed-down" for the appropriate crowd. As an enterprise architect, I don't want to deal with the details of system-level programming and operating system implementations. That is one of the reasons I chose J2EE as my reference framework for implementing applications. Yes, J2EE is an abstraction, a "dumbed-down" framework for me to develop distributed applications.

One of the really neat things about Java is that it's made standard design patterns household names for the developer community. Java is built on standard design patterns and implementation guidelines. Five years ago, the majority of the programmers I talked to didn't know what design patterns were. Now, almost all Java developers are aware of Factories, Listeners, and Adapters. The reason? Core Java objects use the Factory patterns. Observer-Observable and Producer-Listener are a part of the language. Adapters and Proxies are applied all over enterprise Java applications.

J2EE Design

J2EE provides component APIs that are built on solid design principles. However, in order to apply these APIs to build enterprise applications, you need a good design base for developing robust and reusable components. The good news is that years of experience of a multitude of developers are available for other developers to apply, use, and learn from. The best approach is to consult the appropriate documentation and other resources, and not reinvent the wheel for each new application. One of the primary advantages of this approach is that your design paradigm will be supported by a larger community than just the environment that you program in. In some sense, design patterns are simply communication mechanisms for application architects, designers, and developers to share information on best design practices.

J2EE covers a lot of areas. It's impossible for a single person, or a group of developers, to acquire expertise on all of them. This problem is magnified in distributed development environments where communicating the different aspects of design and coordinating the modules of the application becomes an extremely daunting task. Design patterns help alleviate a large portion of this pain.

Sun's J2EE Blueprints provide a great starting point to help developers understand the rationale behind the design of the J2EE platform. The sample Pet Store application helps put this in the perspective of a real-world demo. However, when you design a real-world application, there are other aspects of design that you need to address. With the acceptance of J2EE as the preferred platform for building distributed applications, you'll see richer sources of information on building these applications emerge.

Starting This Month . . .

At ***JDJ***, we want to address this higher level of abstraction. This month and in subsequent issues, you'll see more coverage on applying J2EE into actual applications. In this issue, John Crupi starts a

–continued on page 74

ajit@sys-con.com

AUTHOR BIO

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

J2EE INDEX

J2EE FAQ

WHAT ARE THE COMPONENT TECHNOLOGIES OF THE J2EE ARCHITECTURE?

The J2EE architecture consists of the following types of Java components: Web components, EJBs, application clients, and Java applets. Web components consist of servlets and JavaServer Pages (JSPs).

The component technologies of J2EE facilitate the development of J2EE components. These technologies are offered as a set of distributed system APIs that are specified by Sun via the Java Community Process (JCP) and are open to vendors, including Sun, for implementation. These APIs include :

- ***Enterprise JavaBean (EJB):*** For developing server-side business components
- ***Java Database Connectivity (JDBC):*** For unified RDBMS access
- ***Java Message Service (JMS):*** For distributed messaging
- ***JavaMail:*** For e-mail services
- ***Java Native Directory Interface (JNDI):*** For distributed naming services
- ***JavaServer Pages (JSP):*** For generating dynamic HTML or XML for Web clients
- ***Java Servlet:*** For generating dynamic HTML or XML for Web clients
- ***Java Transaction API (JTA):*** For managing distributed transactions
- ***Remote Method Interface (RMI):*** For distributed object communications

Enterprise JavaBeans are the central theme for J2EE. EJBs comprise Java's middle-tier server-side component model. All the other APIs exist to provide connectivity to and from EJBs.

WHERE DO ALL THE J2EE COMPONENTS EXIST IN A DISTRIBUTED APPLICATION?

J2EE is a true *n*-tier architecture; however, the different components developed on the J2EE framework can be grouped into server-side and client-side components. Applets and application clients are basically pure client-side components and execute in a virtual machine on the client machine. Applets execute in a browser, while applications can execute in other client-side processes.

J2EE offers two types of Web components: servlets and JSPs. The purpose of both types of components is the same: to serve up dynamic content to Web clients. In fact, JSPs are compiled into servlets during execution. This means that JSPs and servlets get requests from Web clients and serve back HTML (or XML). Servlets and JSPs provide the connectivity between the client- and server-side components in a J2EE application. To generate the right response, they can communicate with back-end data sources through JDBC, RMI, JMS, or JavaMail, as well as to middle tier Java components, namely, EJBs. In turn, EJBs can communicate with back-end sources.

EJBs are Java's server-side components. EJB components are developed to model business logic for an application. EJBs can leverage other APIs to connect to and communicate with back-end systems. There are two types of EJBs – session beans, which are associated with the lifetime of a user session, and entity beans, which are associated with server-side business objects that are persisted in a database and live beyond the life of user sessions.

WHAT ARE J2EE CONTAINERS AND WHAT IS THEIR RELATION TO APPLICATION SERVER VENDORS?

A J2EE component needs an execution environment to run in. J2EE containers are runtime hosts for J2EE components. Since there are basically two types of middle-tier J2EE components (Web components and EJBs), there are two types of J2EE containers – Web component containers and EJB containers.

Web component containers are basically servlet engines, JSP engines, and Web containers. A servlet container provides network services for servlet execution, such as support for HTTP, and other request-response protocols. JSP containers are basically servlet containers with the additional functionality of compiling JSP pages into servlets. As mentioned above, JSPs become servlets at runtime. Web containers provide the additional functionality of access to other J2EE APIs, such as RMI and JDBC.

EJB containers provide the runtime life-cycle management environment for EJB components. This includes EJB instantiation, communication, persistence, and transaction management, as well as access to other J2EE APIs, such as RMI and JDBC.

Application server vendors such as WebSphere, BEA, iPlanet, and ATG provide implementations of the J2EE service APIs and the containers in which components built on these APIs can execute. Since there are different types of J2EE containers, there are different J2EE container providers. This means that you can buy the Web and EJB containers from different sources and make them communicate by using standard J2EE services. For example, a JRun servlet engine from Macromedia should be able to work with an EJB container from WebLogic.

J2EE ROADMAP

The Java2 Platform, Enterprise Edition defines the APIs for building enterprise-level applications.

J2SE	v. 1.2
Enterprise JavaBeans API	v. 1.1
Java Servlets	v. 2.2
JavaServer Pages Technology	v. 1.1
JDBC Standard Extension	v. 2.0
Java Naming and Directory Interface API	v. 1.2
RMI/IIOP	v. 1.0
Java Transaction API	v. 1.0
JavaMail API	v. 1.1
Java Messaging Service	v. 1.0

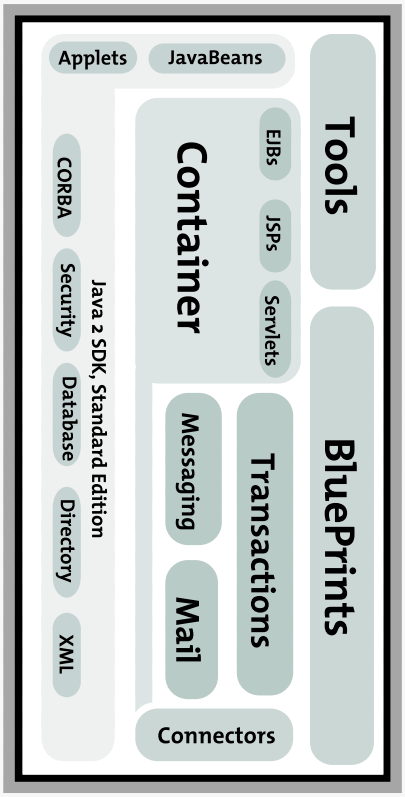
Useful URLs:

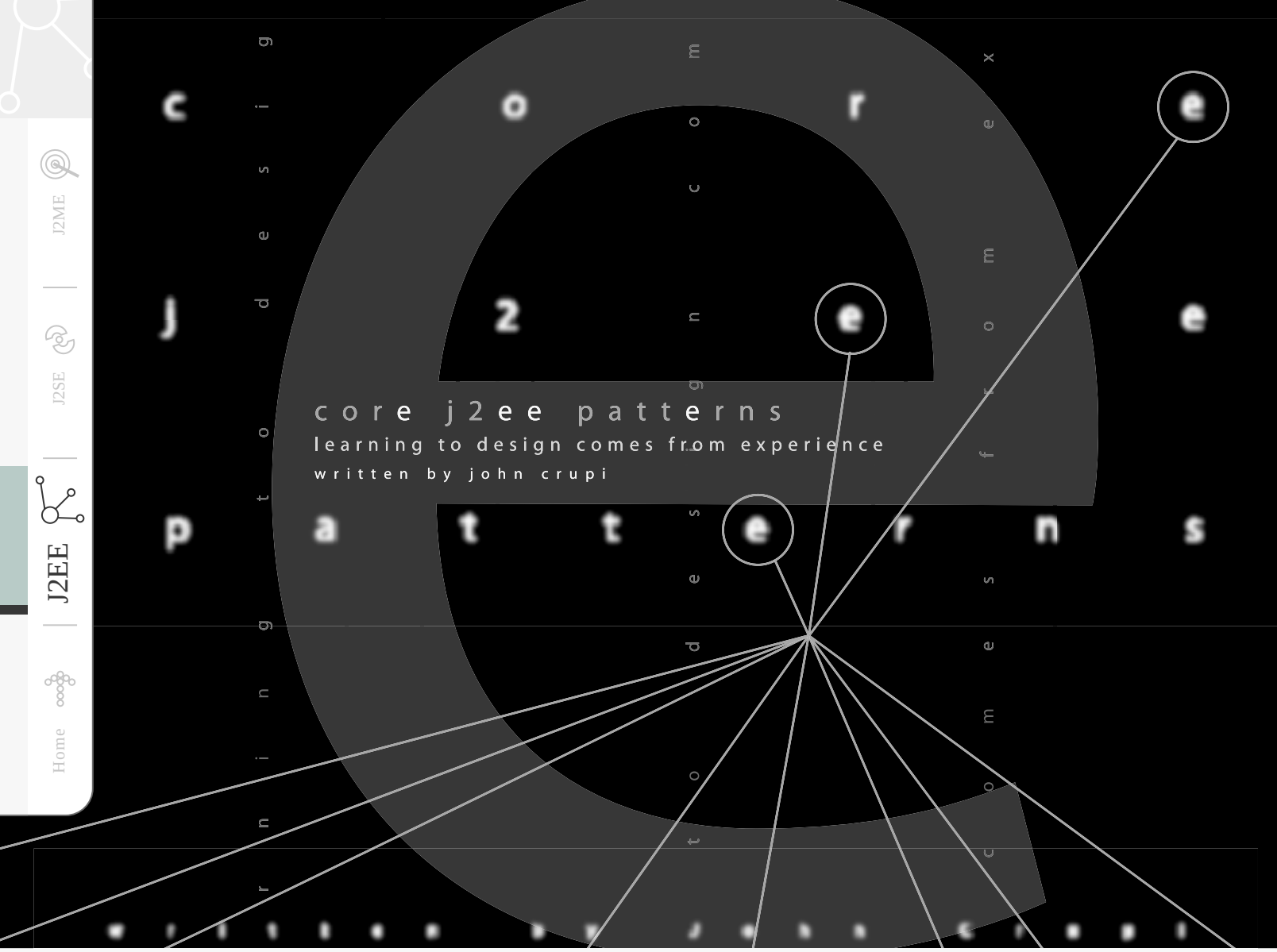
Java 2 Platform Enterprise Edition
http://www.java.sun.com/j2ee/

J2EE Blueprints
http://www.java.sun.com/j2ee/blueprints

J2EE Technology Center
http://developer.java.sun.com/developer/products/j2ee/

J2EE Tutorial
http://java.sun.com/j2ee/tutorial/





welcome to the first installment of the "Core J2EE Patterns" column by the Sun Java Center (www.sun.com/service/sunps/jdc). Every other month, we (Deepak Alur, Danny Malks, myself, and other architects from the Sun Java Center) will discuss various topics from our book, *Core J2EE Patterns, Best Practices and Strategies* (Alur, Crupi, Malks, Prentice Hall/Sun Press, 2001). These topics include each of the 15 J2EE patterns in our catalog, design strategies, bad practices, refactorings and pattern-driven design in the Java 2 Platform, Enterprise Edition (J2EE).

Applying the Technology

Today, as in the past, many of us naively assume that learning a technology is synonymous with learning to design with the technology. Certainly, learning the technology is important for success in designing with the technology. Many existing Java books are excellent at explaining technology details, such as API specifics and so forth, but at the same time they give no insight on applying the technology.

Learning to design comes from experience and from sharing knowledge on best practices and bad practices. The experiences we'll convey in this column are derived from the work we have done in the field. We are part of Sun Microsystems, Inc.'s, Sun Java Center (SJC) consulting organization and have been designing mission-critical J2EE solutions since J2EE technology was introduced.

In our work, we often encounter situations where, because technology is moving so quickly, designers and developers are still struggling to understand the technology, let alone how to design with the technology. It's not good enough to tell designers and developers to write good code, nor is it sufficient to suggest using servlets and JavaServer Pages (JSP) technology for developing the presentation tier and EJB components for developing the business tier.

Since its inception, SJC architects have been working with clients all over the world to successfully design, architect, build, and deploy various types of systems based on Java and J2EE technology and platforms.

Recognizing the need to capture and share proven designs and architectures, we started to document our work on the J2EE platform in the form of patterns in 1999. Although we looked in the existing literature, we couldn't find a complete catalog of patterns that dealt specifically with the J2EE platform. We found many books dealing with one or more of the J2EE technologies, which do an excellent job of explaining the technology and unraveling the nuances of the specifications, but many of these books fall short on design.

After two years of work on capturing J2EE patterns and

documenting them in the proper pattern form, at the JavaOne 2001 Conference we launched our book, which includes the SJC J2EE Pattern Catalog. We also announced that four companies and allies, Rational, TogetherSoft, Forte, and iPlanet, have all agreed to bundle the SJC J2EE Pattern Catalog in their respective tools (http://java.sun.com/pr/2001/06/pr010604-25.html).

This is important in our eyes because patterns lend themselves to tools so nicely. As a matter of fact, the pattern-driven design concept in J2EE technology is difficult without tool support. More about this later.

The Essence of Patterns

First, let's start with a brief overview of patterns – starting with some expert definitions.

In *A Pattern Language,* Christopher Alexander says each pattern is a three-part rule which expresses a relation between a certain context, a problem, and a solution.

Richard Gabriel discusses this definition in more detail in *A Timeless Way of Hacking.* Gabriel offers his own version of Alexander's definition as applied to software: each pattern is a three-part rule that expresses a relation between a certain context, a certain system of forces that occurs repeatedly in that context, and a certain software configuration that allows these forces to resolve themselves.

This is a fairly rigorous definition, but there are also much looser ones. For example, Martin Fowler, in *Analysis Patterns*, offers the following definition: a pattern is an idea that has been useful in one practical context and will probably be useful in others.

As you can see, there are many definitions for a pattern, but they all have a common theme relating to the recurrence of a problem/solution pair in a particular context. Some of the common characteristics of patterns:

- They're observed through experience
- Are typically written in a structure
- Prevent reinventing the wheel
- Exist at different levels of abstraction
- Undergo continuous improvement
- Are reusable artifacts
- Communicate designs and best practices
- Can be used together to solve a larger problem

Categorizing Patterns

Patterns, then, represent expert solutions to recurring problems in a context and thus have been captured at many levels of abstraction and in numerous domains. Numerous categories have been suggested for classifying software patterns, with some of the most common patterns being:

- Design
- Architectural
- Analysis
- Creational
- Structural
- Behavioral

Even within this brief list of categories, we see numerous levels of abstraction and orthogonal classification schemes. Thus, while many taxonomies have been suggested, there is no one right way to document these ideas.

In our J2EE pattern catalog, each pattern hovers somewhere between a design pattern and an architectural pattern while the strategies document portions of each pattern are at a lower level of abstraction. The scheme we have introduced is to classify each pattern within one of the following three logical architectural tiers:

- Presentation
- Business
- Integration

Identifying a Pattern

We have successfully completed and deployed many J2EE technology projects at the Sun Java Center, and over time have noticed that similar problems recur across these projects. We have also seen similar solutions emerge for these problems. While the implementation strategies varied, the overall solutions were quite similar. Let's discuss, in brief, our pattern identification process.

When we see a problem and solution recur, we try to identify and document its characteristics using the pattern tem-

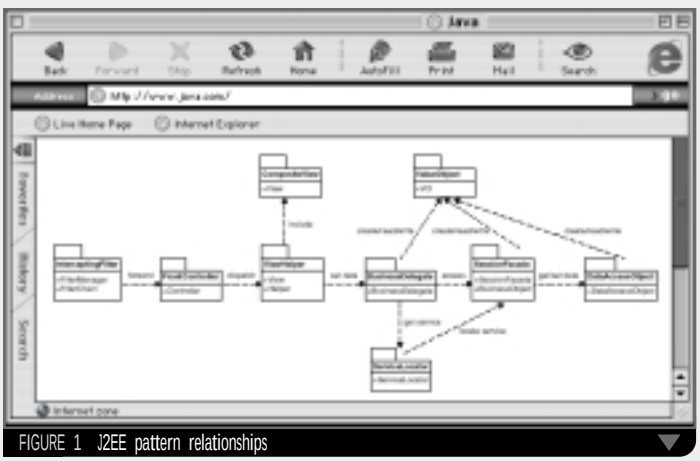


FIGURE 1 J2EE pattern relationships

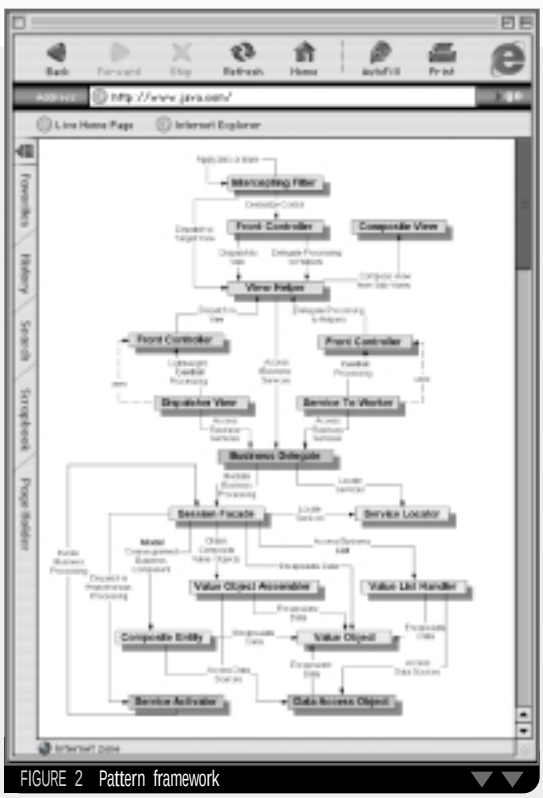


FIGURE 2 Pattern framework

plate. At first, we consider these initial documents to be candidate patterns. However, we don't add candidate patterns to the pattern catalog until we are able to observe and document their usage multiple times on different projects. We also undertake the process of pattern mining by looking for patterns in implemented solutions.

As part of the pattern validation process, we use the Rule of Three, as it is known in the pattern community. This rule is a guide for transitioning a candidate pattern into the pattern catalog.

According to this rule, a solution remains a candidate pattern until it has been verified in at least three different systems. Certainly, there is much room for interpretation with rules such as this, but they help provide a context for pattern identification. Often, similar solutions may represent a single pattern. When deciding how to form the pattern, it's important to consider how to best communicate the solution. Sometimes, a separate name improves communication among developers. If so, then consider documenting two similar solutions as two different patterns. On the other hand, it might be better to communicate the solution by distilling the similar ideas into a pattern/strategy combination.

Patterns vs Strategies

When we started documenting the J2EE patterns, we made the decision to document them at a relatively high level of abstraction. At the same time, each pattern includes various strategies that provide lower-level implementation details. Through the strategies, each pattern documents a solution at multiple levels of abstraction. We could have documented some of these strategies as patterns in their own right; however, we feel that our current template structure most clearly communicates the relationship of the strategies to the higher-level pattern structure in which they are included.

While we continue to have lively debates about converting these strategies to patterns, we have deferred these decisions for now, believing the current documentation to be clear. We have noted some of the issues with respect to the relationship of the strategies to the patterns:

- The patterns exist at a higher level of abstraction than the strategies.
- The patterns include the most recommended or most common implementations as strategies.
- Strategies provide an extensibility point for each pattern.
- Developers discover and invent new ways to implement the patterns, producing new strategies for well-known patterns.
- Strategies promote better communication by providing names for lower-level aspects of a particular solution.

The Tiered Approach

Since this article describes patterns that help you build applications that run on the J2EE platform, and since a J2EE platform (and application) is a multitiered system, we view the system in terms of tiers. A tier is a logical partition of the separation of concerns in the system. Each tier is assigned its unique responsibility in the system. We view each tier as logically separated from one another. Each tier is loosely coupled with the adjacent tier. We represent the whole system as a stack of tiers.

J2EE Patterns

We used the tiered approach to divide the J2EE patterns according to functionality, and our pattern catalog follows this approach. The presentation-tier patterns are related to servlets and JSP technology. The business-tier patterns are related to the EJB technology. The integration-tier patterns are related to the Java Message Service (JMS) and Java Database Connectivity (JDBC) technology.

Table 1, 2, and 3 each list the patterns and a brief description for each tier.

PATTERN NAME	SYNOPSIS
Intercepting Filter	Facilitates preprocessing and post-processing of a request.
View Helper	Provides a centralized controller for managing the handling of a request.
Composite View	Creates an aggregate View from atomic subcomponents
Service To Worker	Combines a Dispatcher component with the Front Controller and View Helper Patterns.
Dispatcher View	Combines a Dispatcher component with the Front Controller and View Helper Patterns, deferring many activities to View processing

TABLE 1 Presentation-tier patterns

PATTERN NAME	SYNOPSIS
Business Delegate	Decouples presentation and service tiers, and provides a facade and proxy interface to the services
Session Facade	Hides business object complexity; centralizes workflow handling.
Value Object	Facilitates data exchange between tiers by reducing network chattiness.
Composite Entity	Hides business object complexity; centralizes workflow handling.
Value Object Assembler	Assembles a composite value object from multiple data sources.
Value List Handler	Manages query execution, results caching, and results processing.
Service Locator	Encapsulates complexity of business service lookup and creation; locates business service factories

TABLE 2 Business-tier patterns

PATTERN NAME	SYNOPSIS
Data Access Object	Abstracts data sources; provides transparent access to data.
Service Activator	Activator Facilitates asynchronous processing for EJB components

TABLE 3 Integration-tier patterns

Using UML

We use UML extensively in the pattern catalog, particularly as follows:

- ***Class diagrams:*** We use the class diagrams to show the structure of the pattern solution and the structure of the implementation strategies. This provides the static view of the solution.
- ***Sequence (or interaction) diagrams:*** We use these diagrams to show the interactions between different participants in a solution or a strategy. This provides the dynamic view of the solution.
- ***Stereotypes:*** We use stereotypes to indicate different types of objects and roles in the class and interaction diagrams.

Each pattern in the pattern catalog includes a class diagram that shows the structure of the solution and a sequence diagram that shows the interactions for the pattern. In addition, patterns with multiple strategies use class and sequence diagrams to explain each strategy.

Patterns Need Patterns

About a year ago, Sun ran a Java technology developer focus group to gather data about tools and patterns. During the pattern section, the question, "Do you think patterns are useful?" was posed to the group. Almost unanimously, the answer was yes. Next, they were asked, "Do you have any problems with using patterns in development?"

The number one problem identified by the developers was that they did not have a good handle on how and when to use patterns together to solve a business problem. I agree. The value in creating a pattern catalog is not just understanding the isolated patterns, but in how the patterns work together. Even more so, what are the patterns that are commonly used together to solve a common business problem?

We created a pattern relationship guide (also known as a pattern language) to visually see how the patterns relate to each other as shown in Figure 1.

In each column we will attept to put each pattern in the context of a business problem and also discuss the patterns that are commonly used together. We call these pattern frameworks.

In Figure 2, we show an example of a pattern framework. Each pattern is represented by a UML package that contains the participants of each pattern. The diagram identifies the end-to-end (across each tier) patterns as can be applied to a business problem.

In Upcoming Issues

I wanted to give you a flavor of what's to come in future columns. We think this will be a great opportunity and forum to exchange and share with you our ideas, vision, and experiences with applying J2EE patterns in the real world. If there are any J2EE pattern topics you want us to address in future columns, please e-mail us at CoreJ2EEPatterns@sun.com.

Author Bio

John Crupi is the chief Java architect of the Sun Java Center. He has more than 15 years of experience in distributed object computing and remains focused on creating reusable, scalable architectures for J2EE technology. He is coauthor of Core J2EE Patterns *(Prentice Hall/Sun Press, 2001) and is currently concentrating on pattern-driven design in J2EE.*

john.crupi@sun.com

Fitting the Pieces into the Enterprise Java Puzzle

Adding functionality with EJBs

Part 3 of 3

WRITTEN BY
TONY LOTON

The story so far: In Part 1 (*JDJ* Vol. 6, issue 4), I covered servlets and gave a practical demonstration of how a basic access control mechanism for intranet applications could be built using Servlet Session Tracking and HTTP Authentication. In Part 2 (Vol. 6, issue 5), I introduced a couple of applets into the architecture and showed how a communication channel could be established between the applets and servlets that comprised the application.

The next version of my application will include the ability for an engineer to transfer tasks to another engineer when he goes on vacation or when he's sick. This added functionality requires a servlet to present the transfer form and subsequently to process the form submission. An Enterprise JavaBean (session) will perform the actual transfer function.

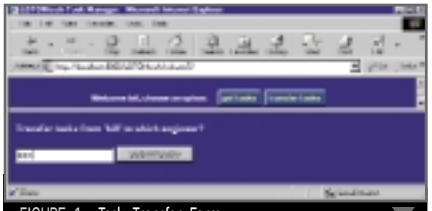


FIGURE 1 Task Transfer Form

AUTHOR BIO
Tony Loton works through his company, LOTONtech Limited (www.lotontech.com), as an independent consultant, course instructor, and technical author. He has a degree in computer science and management and has spent 10 years in IT. His last five years have been devoted almost exclusively to Java, UML, and related technologies.

Task Transfer Servlet

Figure 1 shows what the Task Transfer Form looks like. You can type the name of a recipient engineer and submit the form to initiate the transfer. If the transfer succeeds, the user is presented with a confirmation page (not shown).

I will be using a single servlet to support two functions: presenting the transfer form, and handling the form submission. One servlet, two uses, distinguished by invoking the servlet through its doGet() or doPost() method.

The initial form presentation is performed by the doGet() method of the new servlet, which I've called TransferServlet. Listing 1 provides the code for this method, and for clarity I've delegated the construction of the HTML form to a separate displayForm() function to facilitate the representation of the form in the case of user error. The first time around, the form is displayed with a null error message via the call displayForm(out.user,null).

When the user submits the form, the submission is handled by the same servlet's doPost() method given in Listing 2. This method first gets hold of the engineer to which the transfer should be targeted. If no engineer was specified, it redisplays the form with an error message – displayForm(out,user, "You must specify an engineer") – otherwise the transfer goes ahead courtesy of an Enterprise JavaBean.

Simple EJB Solution

I've created the simplest possible Enterprise JavaBean to perform the transfer of tasks from one engineer to another. The TaskManager EJB is referenced by name via JNDI, and its home interface is obtained. The home interface is used to create an instance of the bean, and this instance is instructed to transfer the tasks. I said that this is the simplest possible EJB, and it is: the remote interface has only one method – transferTasks(…).

The EJB implementation in Listing 3 is a minimal session bean with no transactional attributes, and a single method with a single JDBC statement to manipulate the usertasks database table. I've kept it simple not only for illustration, but also because it's often the best approach. The sole SQL statement succeeds and is committed – or it fails and is rolled back. Either way, transactional integrity is assured.

When presented like this, EJBs look easy, so to spice things up I'll suggest another way of achieving exactly the same functionality.

The Alternative EJB Solution

This time, I'll use an entity bean to represent each individual engineer and I'll rework the TaskManager session bean to coordinate the two entity beans that represent the two engineers. The remote interface for an engineer (entity) bean has the method definitions shown in Listing 4. For any given engineer you can count his tasks, get his tasks, delete them, and assign new ones.

In case you're wondering, the sometimesFail parameter on the assignTasks() method was my way of testing this. If set to true, this parameter causes the method to throw an exception once in a while, which triggers the entire transfer operation to be rolled back.

The TaskManager session bean now has a revised version of the transferTasks method, given in Listing 5. It looks up the two entity beans representing the from- and to- Engineer(s), gets the tasks of the fromEngineer, deletes them, and then adds them one-by-one to the toEngineer.

Adopting this approach introduces many points at which the transactional integrity might be compromised. The deleteTasks call – or any one of the assignTask calls – might fail, leaving the

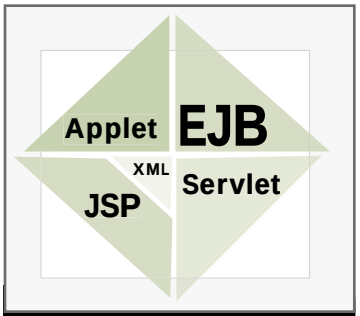


FIGURE 2 Java puzzle

GLOSSARY

API	Application Programming Interface
AWT	Abstract Windows Toolkit
CGI	Common Gateway Interface
CORBA	Common Object Request Broker Architecture
EJB	Enterprise Java Beans
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
J2EE	Java 2 Enterprise Edition
JDBC	Java DataBase Connectivity
JNDI	Java Naming and Directory Interface
JSP	Java Server Pages
RMI	Remote Method Invocation
URL	Uniform Resource Locator
WAR	Web ARchive
XML	eXtensible Markup Language

database in an inconsistent state. Rest assured that you could make it work by setting the transactional properties of the participating session and entity beans, in which case the EJB container will call the ejbLoad and ejbStore methods (assuming bean-managed persistence) on the entity beans at appropriate times to assure the transactional integrity.

This leaves us with (at least) two ways to implement the transferring of tasks between engineers using Enterprise JavaBeans; one way being considerably easier to implement – and more reliable – and the other showing the power and complexity of EJBs. In general, my suggestion would be: beware the consultant who dives straight into the second approach without considering the simpler solution, although, of course, there are times when only a complex solution will do.

Fitting the Pieces into the Puzzle

Having presented the three major architectures offered by the J2EE (the servlet, applet, and EJB) in a way that hopefully shows that these three approaches may be complementary rather than competitive, you might now have some good ideas about the direction in which to take your project. But you might still be a little unsure about the answer to the essential question: Which horse for which course?

I can only give an opinion, but the order in which I have covered these technologies is no accident. For any Internet/intranet application, I would, by default, assume the servlet-only approach initially as providing the thinnest possible clients and the lowest common denominator for compatibility with client Web browsers.

For very limited client-side validation and control, I might stick with the HTML/servlet approach, add some JavaScript, and stop there. In all other situations that necessitated highly interactive or graphical clients, I would use applets with the straightforward applet–servlet communication mechanism based on serialized objects.

In situations that demand true, stateful, remotely referenced server-side objects I would consider Enterprise JavaBeans. I would try to keep my EJBs as simple as possible, and that's where the irony is. The real benefits of the EJB model come through using the standard services for transactions, security, and so on. So if you're not going to use these services – in the name of keeping it simple – you might be better off sticking with simple RMI. In a nutshell, don't treat EJBs as a half-hearted solution; either find an easier method or commit to the EJB approach with all your heart.

I've not yet mentioned CORBA as a candidate for client/server communication or even as an alternative to EJBs. In the time that it took RMI to mature and EJB to take off, CORBA – in the form of the Visigenic (Visibroker) or Iona (Orbix) Java ORB implementations – provided a solution that was more comprehensive than RMI and ahead of EJB. Now that niche has been filled by pure Java technologies and I see CORBA as being limited to the one remaining niche for which it is uniquely suited – legacy integration.

The Three Approaches

I've taken a tour of the three major architecture styles for application development that are available to Enterprise Java developers – namely servlets, applets, and EJB – and I've presented my ideas on how applications may be built comprising all three approaches, with some interesting diversions into areas such as access control. I've fitted the big pieces into the puzzle by offering my advice on which pieces should be inserted first and which later, if at all.

I'll leave you with one final thought. There are a few smaller pieces such as XML and JSP that could be placed into the gaps between the big ones to complete the final puzzle, as shown in Figure 2.

tony@lotontech.com

Listing 1: TransferServlet "doGet" Method

```
public void doGet(HttpServletRequest req
, HttpServletResponse res) throws IOException
{
 // -- get the current http session --
 HttpSession session=req.getSession(true);

 String user=(String)
  session.getAttribute("user");

 if (user==null)
  { /* -- write error message -- */ }
 else
 {
  // -- display the transfer form
  res.setContentType("text/html");
  PrintWriter out = res.getWriter();

  displayForm(out,user,null);
 }
}
```

Listing 2: TransferServlet "doPost" Method

```
public void doPost(HttpServletRequest req
, HttpServletResponse res) throws IOException
{
 // -- get the current http session --
 HttpSession session=req.getSession(true);
 String user=(String) session.getAttribute("user");

 if (user==null)
  { /* -- write error message -- */ }
 else
 {
  // -- handle the transfer form submission --

  res.setContentType("text/html");
  PrintWriter out = res.getWriter();

  String toEngineer=
   req.getParameter("toEngineer");

  if (toEngineer==null)
  {
   displayForm(out,user
    ,"You must specify an engineer");
  }
  else
  {
   out.println("<html>");

   out.println("<head><title>TransferServlet
    </title></head>");

   try
   {
    Context initial = new InitialContext();

    TaskManagerHome taskManagerHome =
     (TaskManagerHome) PortableRemoteObject
```

```
      .narrow(initial.lookup("TaskManager")
      ,TaskManagerHome.class);

     TaskManager taskManager=
      taskManagerHome.create();

     taskManager.transferTasks(user,toEngineer);

     out.println("Transfer to "+toEngineer
      +" succeeded. ");
    }
    catch (TransferException ex1)
    {
     out.println("Transfer to "+toEngineer
      +" failed with an application error. ");
    }
    catch (Exception ex2)
    {
     out.println("Transfer to "+toEngineer
      +" failed with a system error. ");
    }

    out.println("</body>");
    out.println("</html>");
   }
  }
}
```

Listing 3: TaskManager Implementation

```
public class TaskManagerEJB
 implements SessionBean
{
 public void ejbCreate() throws CreateException
  {}

 public void transferTasks(String fromEngineer
  , String toEngineer) throws TransferException
 {
  try
  {
   InitialContext ic = new InitialContext();

   DataSource ds = (DataSource)
    ic.lookup("java:comp/env/jdbc/LOTONtech");

   Connection con = ds.getConnection();
   Statement st=con.createStatement();

   st.executeQuery(
   "UPDATE usertasks SET username='"+toEngineer
   +"' WHERE username='"+fromEngineer+"'");

  }
  catch (Exception e) { /* handle the error */ }
 }

 public TaskManagerEJB() {}
 public void ejbRemove() {}
 public void ejbActivate() {}
 public void ejbPassivate() {}
 public void setSessionContext(SessionContext sc) {}
}
```

Listing 4: Engineer Entity Bean

```
public interface Engineer extends javax.ejb.EJBObject
{
 public int getTaskCount() throws java.rmi.RemoteException;
 public String[] getTasks() throws java.rmi.RemoteException;
 public void deleteTasks() throws java.rmi.RemoteException;
 public void assignTask(String task, boolean sometimesFail)
  throws java.rmi.RemoteException;
 public String getKey() throws java.rmi.RemoteException;
}
```

Listing 5: transferTasks Method with Entity Beans

```
public void transferTasks
 (String fromEngineerID, String toEngineerID)
 throws java.rmi.RemoteException
{
 try
 {
  EngineerHome engineerHome= (IEngineerHome)
  ctx.lookup( "EngineerHome" );

  Engineer fromEngineer=engineerHome
   .findByPrimaryKey(fromEngineerID);

  Engineer toEngineer=engineerHome
   .findByPrimaryKey(toEngineerID);

  String[] tasks=fromEngineer
   .getTasks();

  fromEngineer.deleteTasks();

  for (int i=0; i<tasks.length; i++)
  {
   // Sometimes fail this, to roll back the transaction
   toEngineer.assignTask(tasks[i], true);
  }
 }
 catch (Exception e)
 {
  System.out.println("TaskManager Exception: "+e);
 }
}
```

Download the Code!
www.JavaDevelopersJournal.com

J2EE

Design and Practice

REVIEWED BY AJIT SAGAR
ajit@sys-con.com

Designing Enterprise Applications with the Java 2 Platform, Enterprise Edition
by Nicholas Kassem, Enterprise Team
Addison-Wesley
ISBN: 0-201-70277-0

J2EE Technology in Practice
Rick Cattell, Jim Inscore, Enterprise Partners
Addison-Wesley
ISBN: 0-201-74622-0

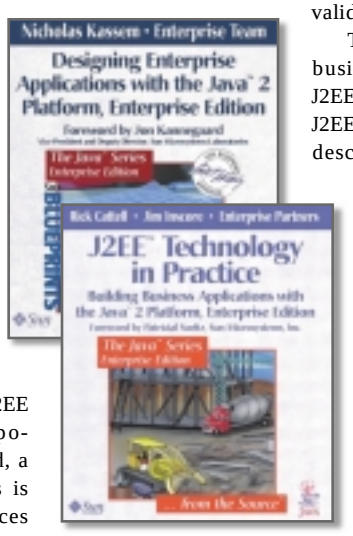
This month I review two books, both of which are valuable sources for developers and architects building enterprise applications using J2EE technologies.

If you're familiar with the J2EE Blueprints from Sun, *Designing Enterprise Applications with the Java2 Platform, Enterprise Edition* is the official "Java Series" book from Addison-Wesley on them. It's a part of the Java Series Enterprise books from Sun. J2EE Blueprints are still available for a free download from Sun's Java site, but if you like to have the professionally bound book, this is it.

J2EE Blue prints were discussed in previous issues of ***JDJ.*** They tie the components of a complex platform with several aspects of software application development into one brief overview. This book is that excellent overview of the J2EE platform. I didn't think it feasible to cover all major aspects of Enterprise Java in 341 pages, but the authors have done a great job in doing just that.

This is a good book for architects and designers. It introduces all the major components of J2EE in a nutshell. Readers may be familiar with some of the information, but I haven't seen another source where it has been covered in such a concise fashion. It introduces the building blocks of J2EE, applicability of individual components in different scenarios, and tiered application development. The book is written by a team of authors from Sun, who have contributed to the development or documentation of the J2EE framework.

The book can be segmented into four basic sections, which may apply to different audiences, or make for different reading sessions. My recommendation is for the reader to go through the whole book. It is not voluminous and does not delve into API details. As mentioned before, it provides an overview and guidelines, not practical application development examples.



Chapters 1 and 2 consist of an introduction to J2EE technologies. The components of J2EE are described, a brief overview of the APIs is given, and the types of services offered by J2EE are described. The discussion on the types of J2EE containers was lucid.

The second part of the book – Chapters 3 to 6 – discusses the basic tiers of applications that can be built on a J2EE base. I found this section particularly interesting. The authors present the business scenarios in which J2EE can be applied. What I liked most was that different types of applications are addressed, but there is no attempt to sell a one-shoe-fits-all concept. The authors make it clear that all aspects of J2EE do not apply to all applications. These chapters help the reader decide what to apply and when.

The next section Chapters 7–9 deals with specific issues that designers and architects encounter when building enterprise applications. The issues related to deploying EJBs, transactions, and security are covered in brief. The last section discusses the Pet Store application – an example application that illustrates the design of a commerce application using the components of J2EE.

I highly recommend this book. It should be useful to a variety of readers, ranging from programmers and architects to product managers. However, it is not a book you can pick up and use as a programming reference. Rather it's a book you can use for design guidelines.

• • •

J2EE Technology in Practice is another book from the Java Series Enterprise Edition set from Addison-Wesley. I picked the book up at JavaOne this year. I was happy to see a book that verifies for the skeptical that J2EE has taken off in the real world.

You may have mixed emotions. I liked the book because it met my expectations – examples of J2EE applications. However, to others it may seem like a series of white papers from different J2EE vendors. If you're looking for development techniques, programming tips, or API details, you won't find them here. This is a book of case studies that validates the J2EE platform.

The book contains examples of business applications built using J2EE technologies. Eight leading J2EE application server vendors describe specific customer applications in which they were able to successfully use J2EE to deploy applications. The applications are in various business domains, including catalog sales, a telecommunications application, and a manufacturing application. Design criteria, issues faced, topology design, transaction management, and many other aspects of applying J2EE technologies are discussed by the vendors. Design patterns, guidelines, and valuable advice are offered by folks who have been in the trenches with the customer.

I recommend this book for readers who are looking at examples of real-world J2EE applications. It is not a comprehensive source, but a good book to help you make a crucial decision before you start a new project using J2EE.

J2EE Application Security Model

A powerful security model that's simple, robust, and reliable

WRITTEN BY
SANJAY MAHAPATRA

The J2EE platform architecture provides for the secure deployment of application components. It emphasizes the declarative approach wherein the application components' security structure, roles, access control, authentication and authorization requirements – as well as the other characteristics pertaining to transactions, persistence, and more – are expressed and managed outside the application code.

J2EE server products provide deployment tools that support the declarative customization of application components for the operational runtime environment. An application component provider isn't expected to implement security services, as it's the responsibility of the J2EE container and server. The security functions provided by the J2EE platform include authentication, access authorization, and secure communication with clients.

Authentication

Authentication is the process by which an entity (such as a user, organization, or program) proves and establishes its identity with the system by supplying authentication data. For users this typically means a user name and password. (In general, authentication data could be comprised of a digital certificate, or even biometric data such as a fingerprint, iris, or retina scan.) A principal is an entity that can be validated by an authentication mechanism; it's identified using a principal name and verified using authentication data.

Authorization and Access Control

Authorization and access control mechanisms ensure that only authenticated principals who have the required permissions to access application components are able to do so. In general, there are two fundamental approaches to controlling access – *capabilities* and *permissions*. The capabilities-based approach focuses on what resources a given user can access. The permissions-based approach on the other hand focuses on which users can access a given resource. The J2EE authorization model uses role-based permissions. A role is a logical grouping of users that's used to define a logical security view for the application. Protected application resources have associated authorization rules – roles that are allowed to access a given component are specified in the application component's deployment descriptor. The deployer maps the roles to actual users using the J2EE server's deployment tools. The J2EE server enforces the prescribed security policies at runtime and ensures that only those users who belong to the appropriate role are able to access the protected components' functionality – while those who don't belong are denied access.

J2EE Containers

A container is part of the J2EE server. It provides deployment and runtime support for application components and is responsible for infrastructural services including security. There are three types of J2EE containers that are meant to house different J2EE application components:

1. ***EJB container:*** Houses EJB components
2. ***Web container:*** Houses servlets, JSPs, static HTML, JPEG files, and more
3. ***J2EE application client container:*** Houses J2EE application clients

Each of these containers has its associated deployment descriptor.

Obtaining the Initiating Security Context

In the J2EE model, secure applications require that the client programs be authenticated. An end user can be authenticated using either a Web or an application client. Once the user is authenticated, an initial security context is generated and maintained by the J2EE platform. This security context is the encapsulation of the identity of the authenticated user principal.

Web Container's Support for Authentication

The <auth-method> element in the Web deployment descriptor is used to configure the type of authentication mechanism such as "BASIC", and "FORM".

```
<auth-method>FORM</auth-method>
```

The deployment tools provided with the J2EE server product insulate us from having to manually write the deployment descriptors; the elements are provided here for reference. The following are some of the authentication mechanisms made available by the Web container and server.

Basic Authentication

HTTP basic authentication is the simplest. When a user attempts to access any protected Web resource, the Web container checks if the user has already been authenticated. If the user hasn't, the browser's built-in login screen is used to solicit the user name and password from the user so that the Web server can perform authentication. If the login fails, the browser's built-in screens and messages will be used. The user name and password are sent using simple base 64 encoding.

Form-Based Authentication

Form-based authentication is used if an application-specific login screen is required, and the browser's built-in authentication screen isn't adequate. The Web deployment descriptor needs to

specify the login form page and the error page to be used with this mechanism. When the user attempts to access any protected Web resources, the Web container checks if the user has already been authenticated. If the user hasn't, the container presents the login form as specified in the deployment descriptor. If authentication fails, the error page, as specified in the deployment descriptor, is displayed.

The <form-login-page> and <form-error-page> elements are respectively used to specify the location of the login and error pages that need to be displayed. Further, the login page must contain fields named precisely j_username and j_password to represent the user name and password, respectively, as shown in Listing 1.

HTTPS Authentication

HTTPS (HTTP over SSL) authentication is a strong authentication mechanism. It requires the user to possess a public key certificate and is ideal for e-commerce applications as well as single sign-ons from within the browser in an enterprise.

Hybrid Authentication

In both the HTTP basic and the form-based authentication, passwords aren't adequately protected for confidentiality. This deficiency can be overcome by running HTTP basic and HTTP form-based authentication mechanisms over SSL. Generally, the use of the CONFIDENTIAL flag in the <transport-guarantee> element of the Web deployment descriptor ensures the use of SSL for data transmission.

```
<transport-guarantee>CONFIDEN-
TIAL</transport-guarantee>
```

All J2EE specification–compliant Web servers support the concept of a single sign-on so that one login session can span multiple applications, thus allowing a user to log in once and access multiple applications.

J2EE Client Container's Support for Authentication

A J2EE application client is a Java application, but differs from a "plain" Java application client in that it's a J2EE component that's deployed in the J2EE "client" container. When a J2EE application client is run, a login window provided by the J2EE application server will pop up and require user name and password fields to be input. Once the user is authenticated, the application will be started. The authenticated user security context is obtained, maintained, and propagated by the J2EE platform so that the client application code doesn't have to deal with the login issue. As with other J2EE components, a J2EE application client is created with the J2EE server's deployment tools. J2EE application clients provide a standard and portable way for authentication, therefore the J2EE architecture encourages the use of J2EE application clients rather than "plain" Java application clients.

Authentication of 'Plain' Java Application Clients

A "plain" Java application client is a nonbrowser based, stand-alone Java application. There's no standard procedure defined by the EJB 1.1 specification regarding their authentication. However, several application servers support a procedure that involves appropriately setting the environmental property constants, java.naming.Context.SECURITY_PRINCIPAL and java.naming.Context.SECURITY_CREDENTIALS, and creating the initial context based on the same (see Listing 2).

> "In general, there are two fundamental approaches to controlling access – capabilities and permissions"

Security Context and Access Control

Once the J2EE platform performs authentication and obtains the security context of an authenticated principal, it's maintained in the background. Whenever an attempt is made to access a protected application component, the container uses the roles known to be associated with the authenticated user in conjunction with the roles authorized to access the component, as prescribed in the deployment descriptor, to either permit or deny access. For example, the container will allow the authenticated user "John Doe," who belongs to the "sales representative" role, to access a protected component with a deployment descriptor setting that specifies "sales representative" as one of the roles authorized to access it.

Web Component Authorization

JSPs, servlets, and HTML pages can be set up as protected resources using J2EE server deployment tools. The <security-constraint> and its subelement <auth-constraint> are used respectively to associate security constraints on Web resources and indicate the user roles that should be permitted access to the Web component. The roles used here should appear in the <security-role-ref> element.

This security context can also be propagated from the Web to the EJB container, which will use it to authorize access to the EJB components' methods.

EJB Component Authorization

The EJB container authorizes access to EJB components based on the caller's security context, in conjunction with the permissions stipulated in the EJB deployment descriptor(ejb-jar.xml), via the <method-permission> and <role-name> elements.

The deployment descriptor setting in Listing 3 grants permissions to all the methods (indicated by the "*") exposed via the home and remote interfaces to an authenticated user in the role of "customer".

The deployment descriptor setting shown in Listing 4 grants specific permissions to the placeOrder() method exposed via the home and remote interfaces to an authenticated user in the role of "customer".

Propagating the Security Context

When one component calls another, the J2EE platform architecture provides for the propagation of the caller security context across components along a call chain. The security context can propagate across the various J2EE containers within the J2EE server. For example, it can propagate from the JSP component housed in the Web container to an EJB component housed in the EJB container. The J2EE platform's support for the propagation of the sensitive security context information provides reliability and convenience. It eliminates the need for the security context information to be passed around as an additional parameter in the business methods.

As such, the deployer can configure the identity selection policy for intercomponent calls so that a specified principal identity other than the original caller

identity will be propagated down the call chain. The proposed EJB 2.0 specification provides a standard technique for addressing the issue of identity selection along a call chain via the <use-caller-identity> and <run-as-specified-identity> elements of the deployment descriptor. If <use-caller-identity> is specified for a component, it results in the propagation of the caller principal along the call chain. If the <run-as-specified-identity> element is used to specify a particular identity, then that specific principal identity is propagated down the call chain and all access control is governed by permissions for the specified identity.

Programmatically Querying the Security Context Information

The getRemoteUser(), getUserPrincipal(), and isUserInRole() methods available in the HttpServletRequest interface provide servlets and JSPs with access to security context information. The getRemoteUser() method obtains the name of the authenticated user, while the getUserPrincipal() returns the principal object associated with the authenticated user. These methods may be useful for recording the user's access to Web components or dynamically generating HTML content that includes the name of the user. Similarly, the isUserInRole(String roleName) queries the underlying security mechanism of the container to determine if the authenticated caller belongs to a given security role. This may be useful for dynamically generating Web content and options based on the role of the user.

Similarly, the EJBContext provides EJB components access into the security context (as well as transaction context). EJBContext provides two methods that allow programmatic access to security-related information, namely getCallerPrincipal() and isCallerInRole(). The getCallerPrincipal() allows the EJB component to obtain the principal object associated with the caller. This may be used for info only in conjunction with declarative management. In a common scenario, getCallerPrincipal() may be called from within an entity EJB component to facilitate saving the loginID of the user who caused the insertion of the row to the database. This can be accomplished by using code in the ejbCreate() method as below:

```
this.loginID =
theEntityContext.getCallerPrincipal
().getName() ;
```

String loginID is a container-managed persistent field that represents the login/user name that caused the insertion of a particular row, and EntityContext theEntityContext was obtained using the callback setEntityContext(). This approach allows sensitive data for recording purposes to be obtained via the EJBContext.

The method isCallerInRole(String roleName) allows the bean implementation to query whether the caller belongs to a particular named role. Typically this is used to provide fine-grained control access and/or programming logic corresponding to that role. When programmatic calls are made to the isCallerInRole() method, the component provider declares the logical role names referenced in the code in the deployment descriptor, and the deployer is responsible for mapping the logical role to an actual security role. This is achieved via the <security-role>, <security-role-ref>, and <role-link> elements of the deployment descriptor.

Future Directions

Currently there's no standard mechanism for the propagation of the security context from the J2EE server to the enterprise information systems (EIS), such as database, ERP, or mainframe transaction processing systems. The J2EE Connector Architecture, which is expected to be included in the next version of the J2EE platform specifications, addresses the propagation of the security context from the application server to the EIS. The objective is to extend the end-to-end security model of J2EE applications to include the EIS tier.

The next release of the J2EE specifications is expected to include the Java Authentication and Authorization Service (JAAS), which implements the Pluggable Authentication Module (PAM) framework and endeavors to make the login services independent of the actual underlying authentication mechanism, so that different security and authentication implementations may be "plugged" in or out seamlessly.

More than one J2EE server product may be used in the production environment of a large enterprise; therefore the passing of the security context between different J2EE server products assumes significance. Currently, the mechanism of passing the security context tends to be specific to the particular J2EE server product. In the current J2EE specification, the EJB to CORBA mapping, which addresses the propagation of the security context over IIOP, is not a required feature. The next release of the J2EE platform specification is expected to make support for the CORBA/IIOP interoperability protocols mandatory, thereby facilitating the seamless passing of the security context between components from different vendors deployed on J2EE servers.

Conclusion

The J2EE architecture provides a powerful security model that's simple yet robust and reliable. It emphasizes the flexible and cost-effective declarative approach of managing security via XML-based deployment descriptors. It provides for the propagation of the security context across components along a call chain. This eliminates the need for the user information to be passed around as an additional parameter in the business method calls, thereby providing reliability and convenience. It also supports the concept to applications of single Web sign-on access.

Author Bio

Sanjay Mahapatra works for Cook Systems International. He's been writing distributed and object-oriented applications for more than five years and is a Sun-certified Java developer and Java 2 platform architect.

smahapatra@cooksys.com

Listing 1

```
<form method="POST" action="j_security_check">
                  <input type="text" name="j_username">
                  <input type="password" name="j_password">
</form>
```

Listing 2

```
  Properties       props = new Properties () ;
  props.put ( Context.SECURITY_PRINCIPAL,       theUserLogin ) ;
  props.put ( Context.SECURITY_CREDENTIALS,     thePassword ) ;
  IntialContext     ic = new InitialContext ( props ) ;
// proceed with lookup  using  above  InitialContext  ic ...
```

Listing 3

```
<method-permission>
  <role-name>customer</role-name>
    <method>
        <ejb-name>CustomerServicesEJB</ejb-name>
        <method-name>*</method-name>
   </method>
</method-permission>
```

Listing 4

```
<method-permission>
   <role-name>customer</role-name>
   <method>
   <ejb-name>CustomerServicesEJB</ejb-name>
   <method-name>placeOrder</method-name>
   </method>
</method-permission>
```

Download the Code! www.JavaDevelopersJournal.com

A J2EE Application Framework Checklist

What to look for to build portable, scalable, and robust applications

WRITTEN BY
STEVEN RANDOLPH

In recent months, there have been significant writings and discussions surrounding J2EE frameworks and the key benefits one provides. I will not spend time reiterating those here. The bottom line is most professionals in this space agree on one thing: application-level reuse is a good thing and the right J2EE framework can deliver just that. This article concerns itself with identifying what specific features to look for in a J2EE application framework.

First, let's decide on an appropriate definition of an application framework.

- **An application framework:** It's an implementation of a set of application-neutral components and services based on the best design patterns, standards, and practices available. These components and services should cover all identifiable tiers of the technology they are addressing. It should encapsulate many application complexities and commonalties, as well as encourage appropriate design and implementation principles of the consumer.

As it relates to J2EE, there already exists a set of well-defined design patterns geared toward the technology. They can be found in the recently published book, *Core J2EE Patterns*. Sun's J2EE Blueprints provide best practice guidelines and architectural recommendations for real-world application scenarios. These enable developers to build portable, scalable, and robust applications. But remember, a framework must provide an implementation of the guidelines and design patterns across all tiers. Before listing what to look for in the implementation, it's important to look at existing framework references.

Struts

This framework implements the Model-View-Controller (MVC) pattern to aid JSP and servlet development efforts. Regardless how appropriate its implementation, without components and services for the other J2EE aspects (EJB, JMS, etc.), and because it supports the presentation tier only, Struts is not qualified to be considered a J2EE application framework.

realMethods Framework

This framework attempts to provide support for all the major J2EE technologies, on all tiers. realMethods contends that its framework is completely design pattern–based and has been developed from the ground up for the past 18 months. There is more on this framework later in the article.

J2EE Blueprints and the Pet Store Demo

Sun's J2EE Blueprints seeks to provide a set of best practice guidelines for J2EE application development. The Pet Store demo represents an application built using these guidelines. It also includes potentially reusable vertical components, such as a shopping cart. Our definition also rules out the Blueprints Pet Store demo since it's an application with an implementation based on design patterns, instead of a core set of reusable classes and services.

What To Look For

There are roughly a half-dozen key features and characteristics to look for in a J2EE application framework. But first, let's define three things a framework is not:

1. ***A library of independent Java components:*** Too often, an enterprise considers itself to have a framework when what it has is a set of reusable components, each not necessarily having any relationship with the set. This isn't to undermine their importance, but such components are often not required for the J2EE application being considered. A framework should provide reusability for all applications across the enterprise using the targeted technology. A calendar component, while reusable, isn't required of every J2EE application.
2. ***An application:*** Even the best designed and written J2EE application is not a framework. Sifting through such an application will hopefully provide some best practices toward proper idioms and techniques of implementation. However, the level of reusability required of a framework would be nearly impossible to extract from the application.
3. ***Any other framework:*** This simply means that you should expect the framework to be constructed "with intention" from the ground up. As software vendors attempt to capitalize on the need for J2EE frameworks, understand the origin of the resulting framework. Be cautious of vendors porting existing frameworks from one technology to J2EE. Ultimately, the best J2EE frameworks will be based on the best design patterns and guidelines related and specific to J2EE.

The Shopping List

Design Patterns Implemented

A framework must have at its core a set of design patterns. Patterns provide

solutions for problems typically encountered in the design and development of J2EE applications. It's important to remember that these patterns should be implemented in such a way as to provide full coverage on all J2EE tiers.

By building your application on top of such a framework, it should immediately inherit the important characteristics intended by the design patterns.

Multiple Tier Support

Any substantial J2EE implementation will require support on the presentation, business, and data/service tiers.

1. ***Presentation tier:*** Encompasses JavaServer Pages, servlets, presentation logic, minimal security, and an object cache
2. ***Business tier:*** Encompasses application business objects, session/entity beans, and JMS technologies
3. ***Data/service tier:*** Encompasses data access considerations as well as asynchronous service definitions and provisions

Extendable

Since a framework is an implementation, it represents a decision toward process definition. To be flexible and useful in as many implementations as possible, it should provide the application designer and developer with design-time and run-time access to both involvement and notification of its execution. This is done by traditional means, such as interface implementation and class extension.

Application Server Neutral

A framework should not make a commitment to any one J2EE application server. None of its core capabilities should have a dependency on a single app server. A feature with a dependency should be peripheral to the essential offerings of the framework, or encapsulated and implemented in such a way that the dependency is transparent to the application.

Configurable

A considerable number of features and services of a framework should be modifiable without making changes to its code. This should happen by means of property and/or XML files. One way in which a framework can maintain application-server neutrality is by allowing property-based configuration. This should simplify migration from one app server to another, or allow a heterogeneous mix of app servers in a single application deployment.

Useful But Not Intrusive

There is a fine line between each. A framework takes a stand with its implementation, but is it forcing your design and development to take place in an unnatural or unacceptable fashion? A framework is of no use if it causes you to retrain in ways that are not transferable to other J2EE efforts (that do not use the same framework). Again, this is where design patterns play a part. By having patterns as the foundation of the framework, you can be assured that the role you will play in consuming the framework will be consistent and predictable.

Code Generation

Since a framework should be based on design patterns and provide multitier support, generating much of the code as it relates to the application and its integration with the framework should be expected. In fact, if this code is not generated, the application runs the risk of incorrectly interpreting the intent of the design patterns already implemented by the framework. This would defeat the purpose of using a framework in the first place.

Dependent (Yet Independent) Services

Just because a framework should be a cohesive implementation of design patterns doesn't mean the application needs to be dependent on all available services. A framework is of the greatest use when an application is able to pick and choose those features that are most important. Loose component coupling by the framework assists in accomplishing this.

Application Server Vendor Positioning

Vendors are busy keeping in step with an ever-changing J2EE specification and JDK. Many have provided vertical components and applications that are mainly tied to their server offering. Most either include, or offer at an additional cost, tools to assist in bean deployment, O/R mapping, and more. Some include Struts or the Blueprints Pet Store demo as part of their download.

But what good are any of these if, fundamentally, the first hurdle is understanding J2EE itself? And the second hurdle, an even higher priority, is to effectively design and implement in J2EE. Fast-moving technology with an increasing level of complexity needs to provide relief in many areas. Frameworks address this need. To date, no application server vendor provides a complete J2EE application framework.

There exists a natural relationship between app server and framework vendors. Considering the neutrality requirement of a J2EE application framework, it should seamlessly complement and enhance any app server vendor offering. A simple comarketing relationship shows that each vendor endorses and supports the other's product. A more "collaborative" relationship is an indicator of a committed, embedded correlation between the two. This type of relationship should serve notice of both vendors' commitment to simplifying and enhancing your J2EE experience.

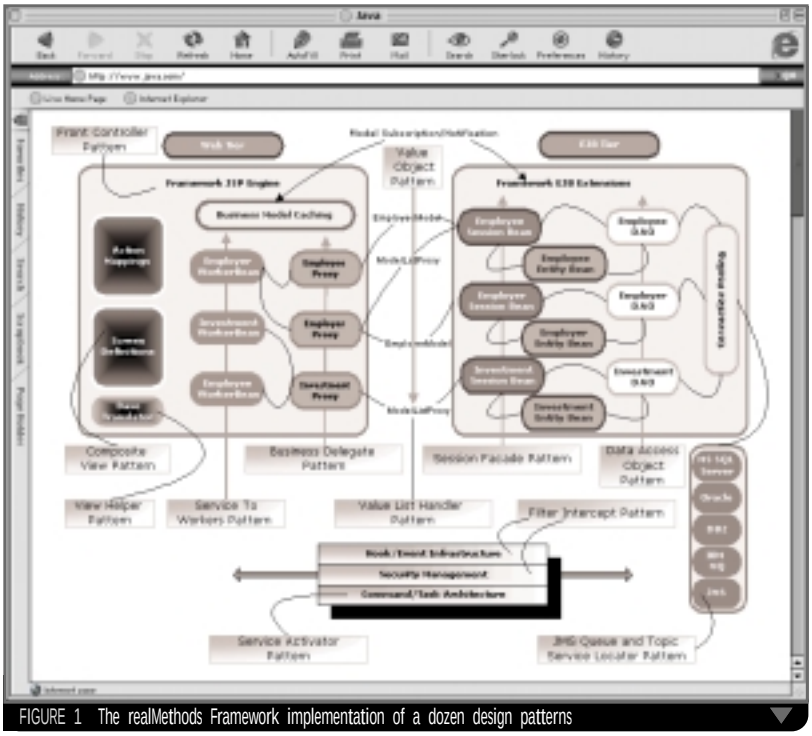


FIGURE 1 The realMethods Framework implementation of a dozen design patterns

> A framework is of the greatest use when an application is able to pick and choose those features that are most important

Author Bio

Steven Randolph is founder and chief technology officer for realMethods, a J2EE software product company based in Bridgewater, Massachusetts. Prior to founding realMethods, he spent more than 12 years designing, managing, and implementing enterprise software solutions for Fortune 500 clients as well as Internet start-ups.

realMethods Framework

All of the necessary design patterns required of any well-structured J2EE application are already implemented by the realMethods Framework. As such, the framework provides support on all tiers (Web, business, and data access). Its main purpose is to provide a cohesive implementation of the best J2EE design patterns. This framework also provides a set of solutions to reoccurring complexities and issues found in most, if not all, J2EE application development.

Observe the realMethods Framework implementation of a dozen design patterns as defined by the book, *Core J2EE Patterns, Best Practices and Design Strategies.*

Unlike single-purpose Java components, the realMethods Framework fulfills the need for application-level reusability across the enterprise (see Figure 1). By furnishing design time, development time, and runtime support, it appropriately allows design and development efforts to be focused on business-related issues, and not necessarily on application infrastructure. Much like the Microsoft Foundation Classes is to the Windows API, the realMethods Framework is to the J2EE Application Server: it applies structure, order, predictability, and simplification toward building J2EE applications.

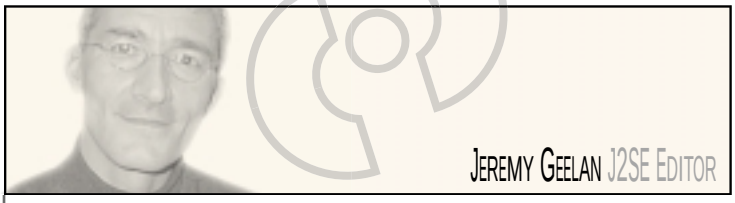
Utilization Is an Issue

Acceptance of J2EE is becoming more widespread, but its utilization remains an issue. Today it is a known fact that many J2EE-based applications remain on the peripherals of the technology, using EJB and JMS technologies sparingly, preferring to focus on the JSP/servlet and JDBC aspects. A framework encourages more thorough consumption of an application server's offerings, and ultimately a deeper commitment by the enterprise to J2EE. As competing technologies attempt to get the attention of software architects and developers, J2EE application frameworks will be an important part of gaining deeper loyalty and dedication to J2EE.

Resources

1. *Struts:* http://jakarta.apache.org/struts/
2. *J2EE Blueprints*: http://java.sun.com/j2ee/blueprints/
3. *Core J2EE Design Patterns:* www.sun.com/service/sunps/jdc/J2EE
4. realMethods Framework: www.realmethods.com; www.j2eeframework.com
5. Alur, D., Crupi, J., and Malks, D. (2001). *Core J2EE Patterns, Best Practices and Design Strategies.* Prentice Hall .

steven@realmethods.com



JEREMY GEELAN J2SE EDITOR

Java Comes of Age

There's an old joke: "It's not progress I'm against, it's just *change* that I loathe!" This isn't one you hear told very often in Internet technology circles!

But Sanjay Sarathy is right. As he says in his "Guest Editorial" at the front of the issue, it would be easy to feel sympathy for the poor CIOs, line-of-business executives, or even developers, who, every time they feel they've figured out how to take full advantage of one edition of Java, find themselves promptly confronted with another.

But for once, or so it seems to me from the sidelines anyway, this isn't a case of technology being dominated by two types of people – those who understand what they do not manage, and those who manage what they do not understand…with Sun Microsystems falling into the second category. On the contrary, the general consensus among industry executives and commentators alike seems to be that Sun – for all its recent headline-making economies – is truly on to a winner with J2SE, J2EE, and J2ME.

Nearly everyone I spoke to or interviewed in the last few weeks and months of conferences and travel – from Mumbai to Seoul to San Francisco to New York to Paris to Copenhagen – has emphasized the self-evident and increasing maturity of the Java platform. Sun, they clearly feel, has been an astute and able custodian of its own technology.

Small Is Big

Everyone has been asking ***JDJ's*** editorial board members, ever since JavaOne, what was truly the biggest news? Well, to me there's little doubt that it was also the *smallest* news: J2ME – the small-footprint Java 2 MicroEdition.

According to Sun's own estimates, there are already 3 million mobile phones enabled with J2ME. Nokia's President, Pekka Ala-Pietila, traveled all the way from Tokyo to San Francisco to announce at JavaOne that Nokia confidently expects this total to increase to a staggering 50 million Java-enabled phones by the end of 2002 and 100 million by the end of 2003.

It's interesting to note that another mobile story has been unfolding...involving not J2ME but J2SE. A new operating system for information appliances, such as advanced PDAs, Web tablets, and so-called "smart phones" has recently been launched. It's based not on J2ME at all, but on plain vanilla Java 2, Standard Edition.

In public beta release for the first time just last month, this new OS supports the full J2SE platform. savaJe XE 0.1.1 (to give it its official name) is the first and only OS to date that allows information appliances to run full J2SE applications, but it probably won't be the last. Anyone who has used a Compaq iPAQ handheld device – reviewed in this issue of ***JDJ*** in the J2ME Section - will tell you: even without going to Japan, it's possible to sense that the next wave of wireless devices is going to embed Java firmly not just into people's minds but also into their handheld devices.

On the other hand, no matter how bright the future may seem for PDAs and for wireless Java, recent events with Psion remind us of the enormous difficulties of being involved with hardware. As the old adage goes, "We are all manufacturers. Making good, making trouble, making excuses." Java can help solve all sorts of technical problems, but those who would use it to achieve ROI still need to have a sound business model.

Conference Draws Near

The JDJEdge 2001 Conference & Expo, the largest Java conference ever held on the East Coast, is fast approaching. Opening keynote speakers include the

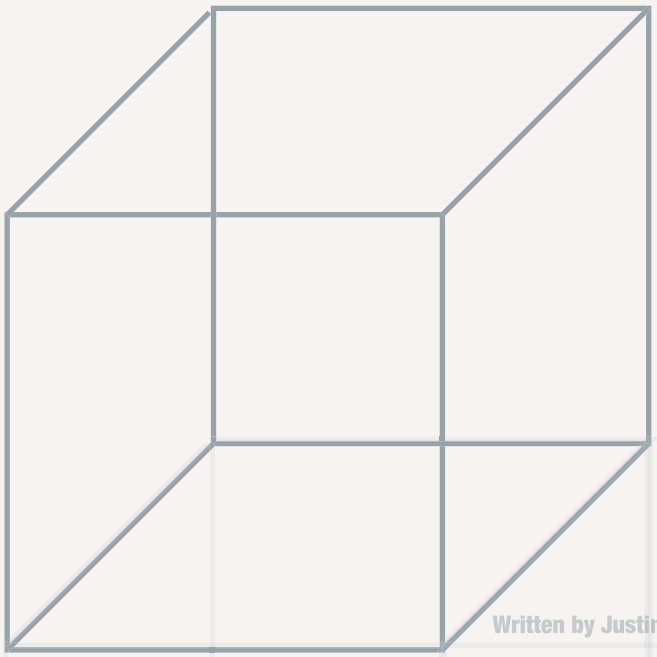
–continued on page 74

jeremy@sys-con.com

AUTHOR BIO

Jeremy Geelan, editorial director of SYS-CON Media, speaks, writes, and broadcasts about the future of Internet technology and about the business strategies appropriate to the convergence of business, i-tech, and the future.

J2SE INDEX



FavoritesComboBox:

A Custom Component for Displaying Recent Selections

Written by Justin Hill and David Lesle

Keep each new class simple and flexible

Swing is a library of graphical components used by Java front-end developers to create robust and functional graphical-user interfaces. Although there are many exciting topics in Swing, one of its most salient features is its ability to be customized.

One of the more powerful and customizable widgets in the Swing library is JComboBox. JComboBox, a combination of a text field and a list, allows the user to select an item from a drop-down list that appears at the user's request. FavoritesComboBox, our extension to JComboBox, is a practical and functional component similar to the Fonts drop-down list in Microsoft Word (see Figure 1).

The concept of FavoritesComboBox is that it provides a listing of the most recently selected items at the top of the list, italicizing and separating them with a horizontal line. A combo box containing a listing of all the countries of the world is an example of a domain in which FavoritesComboBox would be applicable. More often than not, if you live in the United States, "United States" is the selected choice, so why make the user traverse the entire list to find it? FavoritesComboBox solves this problem by placing "United States" at the top of the list.

This article closely examines FavoritesComboBox and its associated classes. In addition to discussing how FavoritesComboBox works internally, the article introduces design patterns, their importance, and how we used them in the design and development of FavoritesComboBox.

FavoritesComboBox

Classes in the Swing package that start with the letter "J" serve an important purpose – they act as the glue between the model and the UI-delegate. In addition, JWhatevers perform "special" operations such as setting up renderers and initializing listeners. FavoritesComboBox (see Listing 1), a subclass of JComboBox, is no exception, and along with performing the default setup specified in the super class, it accomplishes the following:

- Adds a focus listener
- Creates and sets a custom combo box model
- Creates and sets a custom renderer

Before providing a detailed description of the above-mentioned items, we need to clarify some fundamental assumptions regarding user interaction with FavoritesComboBox.

One of the more interesting problems we came across when designing and developing FavoritesComboBox was trying to determine what constituted a selection in the widget. For example, suppose a user chooses an item from the combo box, but the focus remains in the widget. Does this constitute a selection? Furthermore, when a user is keying the up/down arrows to navigate the combo box's list, does this mean the user is making valid selections? We think not. To solve this problem, FavoritesComboBox assumes a selection has not been made until the component's focus is lost. While this basic assumption may seem vague at this point, the following sections will make things clearer.

NestedComboBoxModel

A close look at the pop-up list of possible selections in FavoritesComboBox shows two separate user-selectable lists. We've separated them into distinct ComboBoxModels – one with recent selections and one with all the valid choices. A FavoritesComboBoxModel combines these two models into one for presentation in FavoritesComboBox. The underlying behavior that supports this combination of models into one is provided by the base class – NestedComboBoxModel.

We developed NestedComboBoxModel as a generic component that takes the contents of any number of individual ComboBoxModel instances and presents them as a single concatenated list of items. Support for the basic features of a ComboBoxModel is inherited from its base class,

AbstractComboBoxModel. This provides the initial required interfaces and support for registering event listeners and firing events to them. We had to implement the remaining methods of the ComboBoxModel interface in order to present the combination of several submodels as a single list of elements.

To report the current selection the model must keep track of the currently selected submodel. If one is defined, getSelectedItem() delegates the request to the selected submodel. For applying new selections the model must search through all elements to find the submodel that contains the new selection. Again, the operation is delegated to the submodel, and the new submodel is recorded as the currently selected one – after the previously selected submodel's selected item is cleared. This ensures that if a submodel is presented in multiple visual components, the selections are consistent.

NestedComboBoxModel, as we've shown, is a robust ComboBoxModel capable of grouping distinct ComboBoxModels into one entity and is an important part of the success of FavoritesComboBox.

FavoritesComboBoxModel and FavoritesRecentComboBoxModel

FavoritesComboBoxModel extends NestedComboBoxModel, thus it combines recent selections with possible selections from two separate ComboBoxModels. One important customer requirement of our component was the ability to share recent lists across similar combo boxes. As a result we introduced a facility for models containing related items to share the list of recent selections. This is possible by assigning a type (of class String) to each instance. Combo boxes that share the same type will adjust their set of recent selections when any other combo box of that type changes selection. Due to the interesting communication interaction between recent lists, we've chosen to delegate this functionality into a separate model – FavoritesRecentComboBoxModel.

To construct a FavoritesComboBoxModel, a programmer must pass an existing ComboBoxModel into the constructor. Internally, the passed-in model will be used to define the "whole" list. In addition, FavoritesComboBoxModel will create another model, FavoritesRecentComboBoxModel, to be used as the recent list. Finally, both models will be added as submodels to FavoritesComboBoxModel.

The "shared" recent list behavior is possible because all the FavoritesRecentComboBoxModels are notified when a change occurs in a similar combo box. This event notification is implemented in the FavoritesComboBox component through the use of the Observer design pattern (described in detail in the Design Pattern section).

Along with handling shared recent lists, FavoritesRecentComboBoxModel has several other obligations. One of the peculiarities of JComboBox is that it has a getSelectedIndex() method. However, the ComboBoxModel can't report the current selection by index, so JComboBox scans the model to determine the selected position when showing the pop-up. JComboBox's implementation means each value must be unique as determined by equals(). However, this paradigm won't work for our component since the same item can exist in both submodels. To overcome this restriction, all items in FavoritesRecentComboBoxModel are encapsulated in a wrapper. The two important methods in our Wrapper class (see Listing 2) are toString(), which delegates the call to the item wrapped, and extendedEquals(), which provides a less restrictive comparison that's needed during focus-gained and recent-model update.

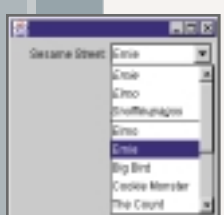


FIGURE 1 A FavoriteComboBox

While Wrapper solves our selection index and equality problem, we use another approach to obtain the combo box's selected item. We now have a FavoritesComboBoxModel that can have selected items that are either Wrappers or the "actual" objects. However, we want FavoritesComboBox getSelectedItem() to report the "actual" object. In the case where the selected item is in the recent list, we want to return its corresponding "actual" object. To achieve this behavior, getSelectedItem() delegates the responsibility to the FavoritesRecentComboBoxModel's getCurrentSelection(). The getCurrentSelection() method determines the selected item [via the model's getSelectedItem()] and, in the case of a Wrapper, returns the encapsulated "actual" object; otherwise, it returns the selected item – the "actual" object.

FavoritesListCellRenderer

To display the most recently selected items at the top of the list, italicized and separated by a horizontal line, we needed to create a custom ListCellRenderer. Our class, FavoritesListCellRenderer, extends DefaultListCellRenderer and is capable of determining which items exist in the recent list as well as where the horizontal divider should be placed. With most ListCellRenderers the rendering logic is defined in the getListCellRendererComponent() method; our renderer is no exception. The following code fragment contains the logic for altering the item's font and displaying the underline:

```
if(anIndex < recentModel.getSize())
  label.setFont(this.recentListFont);

if (anIndex >= 0 && anIndex == recentModel.getSize()-
1)
  label.setBorder(this.underlineBorder);
```

As you can see by the code fragment, the font change and underline placement are contingent on the size of the recent model.

FavoritesComboBoxFocusAdapter

FavoritesComboBoxFocusAdapter, a subclass of FocusAdapter, is added as a focus listener to receive focus events from the combo box component. Focus events are fired internally when a component either gains or loses focus. Selections in FavoritesComboBox are not made until the component's focus is lost.

As we alluded to earlier, JComboBox and its associated default model implementation internally set the current selection whenever items in the combo box change. One example of this behavior is that as a user navigates the drop-down list via the up/down keys, the model changes the current selections. Although this selection behavior is acceptable for JComboBox, it's not adequate for FavoritesComboBox to use the same selection paradigm.

To handle the selection properly, we determined that an item in the combo box is selectable only when the component loses focus. If we had chosen to follow JComboBox's selection paradigm, then in the case of user navigation via the keyboard, we would be populating the recent list whenever the user pressed the up/down keys – which makes no sense when you think about it. In our selection paradigm the recent list is updated only when the combo box component loses focus. The following sections describe in detail how FavoritesComboBox handles the two types of focus events – focus gained as well as focus lost.

public void focusLost(FocusEvent e)

The focusLost() method is defined in FocusListener and is invoked when a component loses the keyboard focus; in our case, when the combo box component loses focus. Although the code contained in focusLost() is minute, it serves a powerful purpose – updating the recent list. The following is focusLost()'s implementation:

```
if (!(aFocusEvent.isTemporary())) {
   JComboBox source =
     (JComboBox)aFocusEvent.getSource();
   Object selectedItem = source.getSelectedItem();
   if (selectedItem != null)

favoritesModel.getRecentModel().updateRecentModel
   (selectedItem);
  }
```

The first line of code simply ensures that the focus has not been lost temporarily. The isTemporary() method allows an application programmer to differentiate between a focus shift between components (returns false) and focus events, which are triggered via window deactivated, window activated, and more (returns true). Once we've determined the focus lost is not temporary, we can invoke FavoritesComboBoxModel's updateRecentModel() method, passing in the selected item from the combo box as a parameter. The updateRecentModel() method simply updates its recent list along with firing an event that notifies all registered listeners. The event-firing mechanism (see Design Pattern section) enables the recent list to be shared for all combo boxes of the same type.

public void focusGained(FocusEvent e)

The focusGained() method is defined in FocusListener and is invoked when a component gains focus; once again, in our case, when the combo box component gains focus. Initially we decided to use focus events for handling combo box selection and focusLost() exclusively; as a result, focusGained() had an empty implementation. However, during initial development we discovered a substantial problem and consequently found a useful purpose for focusGained().

The problem we encountered can best be explained through example. Suppose a combo box exists that contains a listing of all the universities in North America in alphabetical order. This combo box would be large and, for argument's sake, let's assume that St. Peter's College is the current selection. When the combo box gains focus and the drop-down list is displayed, the list displays the current selected item (St. Peter's College). Unfortunately, in this example our recent list, residing at the top of the drop-down list, won't be displayed since the visible portion of the list is centered on St. Peter's near the bottom. To solve this problem, we took advantage of focusGained() by simply obtaining the combo box's selected item, finding its match in the recent list, and setting the selected item to the recent list's corresponding item, guaranteeing that the recent list will always be visible when the drop-down list is displayed. As with focusLost(), the focusGained() implementation is fairly straightforward and is as follows:

```
JComboBox source = (JComboBox)aFocusEvent.getSource();
if (source != null)
  return;
this.favoritesModel.ensureRecentSelectedOnFocusGained(
source.getSelectedItem());
```

The ensureRecentSelectedOnFocusGained() method uses our extended equality check to search the recent list for the match and, if found, selects it, ensuring recent list visibility.

Design Patterns

Design patterns are reusable solutions to common problems that arise during software design.

With the publication of *Design Patterns* in 1995, several of these solutions were cataloged and named. One of the many benefits of design patterns is that they enable software developers to abstractly discuss designing and developing software in a common vernacular. During the design of the FavoritesComboBox component, we relied on several design patterns to guide the structure of code and ensure flexibility. One of the more prominent patterns we used, the Observer, played a critical role in maintaining the recent list in FavoritesRecentComboBoxModel.

In *Patterns in Java*, Grand says the Observer design pattern "allows objects to dynamically register dependencies between objects, so that an object will notify those objects that are dependent on it when its state changes."

The roles of the Observer are:

- ***ObserverIF:*** This interface defines a method, which is invoked upon notification. In our component, FavoritesModelChangeListener plays this role.
- ***Observer:*** This class is a concrete implementation of ObserverIF. Our concrete implementation of FavoritesModelChangeListener is FavoritesRecentComboBoxModel.
- ***Observable:*** This class is responsible for maintaining a list of registered Observers and delivering notifications. FavoritesRecentComboBoxModel performs this role.

Our Observer implementation doesn't include as many classes as the "classic" Observer pattern. The reason we have fewer classes is because, for simplicity, we've chosen to ignore the ObservableIF role and have FavoritesRecentComboBoxModel play the role of both Observer and Observable. To see how these classes interact along with the running of our test application, the FavoritesComboBox component is available on the ***JDJ*** Web site.

When a selection is made in a FavoritesComboBox, it updates its recent model. In addition, it notifies all registered FavoritesModelChangeListeners of the event. When each listener – in this case other FavoritesRecentComboBoxModels – receives notification, the selected item is inserted into its recent list, providing us with the desired functionality of "shared" recent lists. In addition to employing the Observer design pattern, our component uses the Factory Method (creating the renderer) and the Adapter (FavoritesComboBoxFocusAdapter) design patterns.

Future Enhancements

FavoritesComboBox has been a useful widget in our library of GUI components. However, as with many components in their infancy, there are always features left unimplemented. The following two features would prove useful to FavoritesComboBox:

1. ***Recent List Persistence***

Currently, when an application containing FavoritesComboBoxes is launched, the recent list is empty; however, it would be nice if the combo box was populated with its last known recent items. This could be accomplished in several ways. One way to persist the recent list, albeit a simplistic approach, would be through the use of Java Serialization.

2. *Compound Renderer Support*

To change the font of the recent list and create a visual cue (the separator) between the recent list and the "whole" list, a special renderer, FavoritesListCellRenderer, was implemented. In many cases, a developer will want to use another renderer with FavoritesComboBox, but will be unable to since JComboBox supports only one renderer. For example, suppose a developer created a combo box containing a list of the universities in the Big Ten conference. The developer might want to "spice up" the combo box by providing icons displaying the school's mascot and/or colors. If the developer wanted to make this combo box a FavoritesComboBox, he or she would be unable to render both the icons and the separator/italics. As a result, FavoritesListCellRenderer could be altered to take a ListCellRenderer as one of its constructor parameters. Furthermore, FavoritesListCellRenderer's getListCellRendererComponent() method would need to contain additional logic in order to compound the two separate renderers into one.

Conclusion

FavoritesComboBox and its associated support classes demonstrate how a useful component can be created through extending Swing classes and taking advantage of the flexibility of JComboBox. In our component each aspect of our enhancement to the basic Swing functionality was assigned to a separate custom class – keeping each new class simple and flexible.

During design and development we leveraged well-known design patterns in order to increase the reliability and maintainability of our custom component. In addition, new requirements, including persistence and custom renderers, were discovered during FavoritesComboBox use in real applications. Although the design of GUI components can be quite complex, we're confident that these, along with other future enhancements, can be accomplished due to the component's good object-oriented design and use of design patterns.

Acknowledgment

We would like to thank the ASCltd team for their careful review and thoughtful suggestions.

Resources

1. Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley.
2. Grand, M. (1998). *Patterns in Java*. Vol. 1. Wiley.
3. Eckstein, R., Loy, M., and Wood, D. (1998). *Java Swing*. O'Reilly & Associates.

Author Bios

Justin Hill, a Sun-certified Java programmer, works for Advanced Systems Consulting, a Java-centric consulting firm in Chicago. He's currently working on a Java-based source control management front-end to be released to the open-source community.

David Lesle, a Sun-certified Java programmer, works for Advanced Systems Consulting. He's currently designing and developing "front office" multitier Java applications at a Chicago-based trading firm.

jhill@chicagojava.com

dlesle@chicagojava.com

Listing 1

```
package com.chicagojava.awp.client.components.combobox;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;

/***********************************************
<code>FavoritesComboBox</code> is a combobox widget that visually
displays two separate lists: 1) the entire list of selections and
2) a list of favorite (or more recently selected) choices. This
combobox widget mimics the behavior of the "Font List" combobox in
Microsoft Word.
***********************************************/
public class FavoritesComboBox extends JComboBox
{
  /***********************************************
Constructs the <code>FavoritesComboBox</code> and sets its associated model. @param aModel the combobox's model
  ***********************************************/
  public FavoritesComboBox(FavoritesComboBoxModel
    aModel)
  {
    super();
    setRenderer(createFavoritesComboBoxRenderer());
    addFocusListener(
      new FavoritesComboBoxFocusAdapter(aModel));
    setModel(aModel);
  }

  /***********************************************
Sets the maximum number of items the most recent favorites list
can hold. @param aMaximumNumber the maximum number of items in the
list
  ***********************************************/
  public void setMaximumNumberOfFavorites(int
    aMaximumNumber)
  {
    ((FavoritesComboBoxModel)
      getModel()).setMaximumNumberOfFavorites(
      aMaximumNumber);
  }

  /***********************************************
Returns the maximum number of items the most recent favorites list
can hold. @return the maximum number of selections allowed in
recent list
  ***********************************************/
  public int getMaximumNumberOfFavorites()
  {
    return ((FavoritesComboBoxModel)
      getModel()).getMaximumNumberOfFavorites();
  }

  /***********************************************
Returns the currently selected item. This method is overridden to
invoke the <code> FavoritesComboBoxModel's getCurrentSelection()
</code> method. @return the currently selected list object from
the data model
  ***********************************************/
  public Object getSelectedItem()
  {
    return ((FavoritesComboBoxModel)
      getModel()).getCurrentSelection();
  }

  /***********************************************
A factory method that creates the default renderer to be used for
the combobox. Subclasses should override this method in order to
provide their own renderer. @return the default list cell renderer
  ***********************************************/
  protected ListCellRenderer
    createFavoritesComboBoxRenderer()
  {
    return new FavoritesListCellRenderer();
  }
}
```

Listing 2

```
 //a static inner class found in
//FavoritesRecentComboBoxModel
protected static class Wrapper
    implements Serializable
{
  private Object wrappee;

  public Wrapper(Object aWrappee)
  {
    this.wrappee = aWrappee;
  }

  public boolean extendedEquals( Object o )
  {
    return equals( o )
      || wrappee.equals( o )
      || (o instanceof Wrapper &&
      wrappee.equals( ((Wrapper)o).wrappee) );
  }

  public String toString()
  {
    return wrappee.toString();
  }
}
```

Download the Code! www.JavaDevelopersJournal.com

James Gosling on...

Real-world Java

On a late sunny Tuesday afternoon, James Gosling, the creator and father of Java, takes time out to chat with JDJ's Alan Williamson and Blair Wyman.

<williamson>: **How are you finding JavaOne so far?**
<gosling>: There's an awful lot of energy here and just seeing what people are up to is a lot of fun. As we go from year to year, things are moving at such a quick pace. Seeing how much stuff is incredibly real these days is quite a rush.

<williamson>: **Are you still involved with Java?**
<gosling>: It's my job, every day.

<williamson>: **We can't imagine you sitting there in front of a compiler; do you?**
<gosling>: Actually for the last couple of months I've been tormenting the guy who owns the compiler source and I've been hacking on it relatively heavily. But I think someday he will forgive me.

<williamson>: **What's the big surprise this year when you walk around all the booths and exhibitors?**
<gosling>: The thing that I get the biggest kick out of is all the stuff that is now in shrink-wrap

> “There’s now such a high threshold for a major announcement”

boxes. When we did that goofy shopping videotape, none of those devices were fakes; none of them were balsa blocks. They were all prototypes. They were all shipping in volume shrink-wrap gizmos; it's just terribly real. A year or two ago when we were talking about these devices, it was of academic interest to most people there. To the people who were actually building it, it was very interesting.

Today it's a completely different game because anybody here at the forum can really enjoy the feedback. It's nice to have the U.S. headed out of being a third-world country in terms of its phone system – you can go to a phone store in San Francisco and buy a Nextel phone. You can go to the Motorola Web site and get the developer kit. There's a little handshake you have to do, but it's relatively minor and you too can hack your phone set, you too can build an app that does anything from video games to enterprise data stuff that goes end-to-end.

<williamson>: **With respect to the whole spectrum of Java, it's now covering a tremendous amount of space there. Which particular area is now ringing your bell?**
<gosling>: I tend to be a science geek, and for the longest time I was Mr. Everything Except Enterprise. All kinds of things ring my bell in that area. A while ago I visited the Keck Observatory at the top of Mauna Kea and saw how Java has really taken over. The giant telescope crowd and the tools they're building are just incredible.

My first paying job was working for a group of physicists writing satellite data acquisition software for the Isis 2 satellite – that was a real rush. Seeing people doing this kind of stuff is pretty amazing.

<williamson>: **Java is a wonderful language; it's your child, I think people would agree. I'm just wondering, are there still any warts on your child that you'd like to see removed? Are there any problems in Java that you think need fixing or changing?**
<gosling>: The answer is yes and no. In some sense people are being very successful with it. In another sense, if I had a clean slate and was doing things differently, there are many things that would come out differently. I actually have a small Web site that's this long laundry list of things that would be entertaining. It goes from, yeah that might be worth doing to that's truly goofy. A lot of these things are pretty hard design trade-offs. A lot of engineering is not a black-and-white choice – this is the right thing to do and that's the wrong thing. It's similar to Whackamole, a game you play on the Santa Cruz boardwalk and various other places. There's a big board with a little mechanical mole that sticks his head out of the ground and you have a big baseball bat that you whack it down with, but he pops out somewhere else.

A lot of program design (and any other kind of engineering) is like that. For instance, many things about the basic notion of a class-based system in which you have subclassing and inheritance of implementation have issues. You can get around some of that with different styles.

Actually Josh Watt came out with this book called *Effective Programming*. It contains information on how to think about object-oriented programming and how to avoid some of the pitfalls we've discussed over the years. For example, one of the solutions that often comes up is: throw away object-oriented computing completely and go for something along the lines of delegation models that some people have used – it's similar to class inheritance but somewhat different. It solves some problems, but creates its own. There are things to be uneasy about but no clear answers. You could do all kinds of interesting experimentation.

<wyman>: **You spoke of being a science geek. Do you see Java playing in that space? Do you see it compete against FORTRAN, or what was your satellite acquisition software written in?**
<gosling>: PDP8 assembly code.

<wyman>: **PDP assembler, wow!**
<gosling>: That dates me, right? A PDP8 has less compute power than your average smartcard.

<wyman>: **Is there much PDP8 code in Java?**
<gosling>: There's some learning from writing PDP8 code. I had been programming for a number of years before I found a machine beefy enough to run FORTRAN. By then I was writing CDC6000, 7000 assembly code. The cyber series and its predecessors were pretty hot. It was almost like half an MIP.

<wyman>: **MIP is meaningless information for product salespeople?**
<gosling>: Yeah, exactly. ;-)

<wyman>: **In the science arena, do you see the megahertz catching up to make Java effective in a real-time programming environment?**
<gosling>: It's been effective in real time for quite a while. Lots of people have been real-time programming in Java for years. The issue has never been megahertz and speed. If you talk to real-time people, they hardly care about speed; what they care about is determinism, namely, when it's

> "If you talk to real-time people, they hardly care about speed; what they care about is determinism, namely, when it's time to adjust the flutter on the F16 wingtab, I want to do it now please"

time to adjust the flutter on the F16 wingtab, I want to do it now please.

<wyman>: **So no GC cycle then? ;-)**
<gosling>: No, no waiting for the garbage collector to be done with its business. No waiting for the paging system. No waiting for the kernel to context-switch out of the sendmail daemon. When you've got to pull the cadmium rods out of the reactor core, you have to do it now.

<williamson>: **One thing that we've been speaking to all the vendors about, and one thing they've said is that it's nice to see that there are no major new announcements with respect to Java. It's as if Java is maturing to a state where we're now actually using it, and we don't need to help it any longer. In that respect it's cool to see Java now reaching in. Where do you think Java is going in the next 12 months?**
<gosling>: That comment about no major announcement feels kind of weird because when I listen to some of the things that are going on, maybe one of them, four or five years ago, would have counted as a major announcement. There's now such a high threshold for a major announcement.

<williamson>: **Seriously impressive type of situation going on now.**
<gosling>: Yeah, it's like, "Oh gee, Mr. Nokia, only 100-million cell phones, that's boring." We've gotten jaded these days.

<williamson>: **Isn't that a sign of the language maturing though?**
<gosling>: Well, it's sort of maturing and engaging and it's a community thing. We've been trying to get to the point where Sun is not the driver of Java, and I think we've been pretty successful at it. The really cool stuff in Java is not happening at Sun. We haven't been slowing down in what we're doing; the rest of the world has been ramping up. Would class, as a really huge announcement, be like the uptake in MID-P?

What people are doing with development tools, the Java faces thing, there's a way to do a UI toolkit that's supported by IDE tools, yet projects its UI across the Web. That sounds to me like a pretty major announcement, but it's not us doing it. It's not guys with Sun badges doing most of the design and the work. It's people from Borland, WebGain, etc., who are doing it.

<williamson>: **How do you feel about the overall issue that's often debated in newsgroups about open sourcing Java?**
<gosling>: If I actually knew what people meant by open sourcing, I might be able to answer the question. In a strong sense it's been open sourcing from the beginning. We've always shipped the source; anybody who wants it can easily get hold of it. The thing that blocks the zealots calling it open sourcing is that there's actually something that we care about: if somebody has a Java program, and somebody else has something that they call a Java platform, the program ought to run. So we get uptight about that particular point. On average, developers actually care about that, they seem to think that that's a good thing. Yet the open-source community pillories us. Our license is identical to any of the other open-source licenses, but we have these catches that are mostly about interoperability.

<williamson>: **Another hot question we get asked is, back in December 1990, it was released as Java2, but it was actually 1.2. Now it's going to 1.3. Where does this Java2 come into it?**

> "In a strong sense, it's been open sourcing from the beginning. We've always shipped the source; anybody who wants it can easily get hold of it"

<gosling>: Marketing guys, what can I say! 1.2, 1.3, 1.31, that's the numbering scheme in the source-code system; that's the numbering scheme that the engineers actually believe.

<williamson>: **Will we ever see it bump into 2.0?**
<gosling>: Beats the crap out of me.

<williamson>: **I heard 1.4 is around the corner at the end of the year. Then I heard rumors that Tiger 1.5 is now starting to have some serious development**.
<gosling>: I've been working on the Tiger complement for several months now.

<williamson>: **Anything you can tell us about that?**
<gosling>: Nothing useful. On the piece I've been working on are applications, other than the compiler, that would like to use the guts of the compiler. I got into this because I was building a tool that could play with the animated graph. I had built myself yet another Java compiler clone because I wanted to get at the syntax tree, but that seemed kind of goofy because the compiler we had is nicely structured. All it needed was for its innards to be exposed as an API. What I've been working on for the last couple of months is cleaning up the guts of the compiler, so it could be an accessible API.

<wyman>: **You speak of the JavaC compiler and expose it at an API level, or is it too soon to talk about it? Would you be able to give me the parse tree and show me the nodes and all the goodies?**
<gosling>: Yeah, that's what it does now. There's this question of what to do with it. We got started on it because that compiler is officially a Sun product and we needed it. The question is whether to just sail on doing that or start a JCP group to figure out what to do. We've got a first cut at an API, but we'll have to talk to people to

find out what kind of interest there is. For me, I'm doing it because I needed it, whether it turns into part of the product is another question.

<wyman>: **Do you see any changes to the Java Virtual Machine specification underneath the Java language? To extend any of the architectural limits that exist in the byte codes?**

<gosling>: There are actually remarkably few limits. We have this long laundry list of things that would be cool to do, most at the language-level. It's interesting to see how few of them have any impact on the virtual machine. The number one and two language requests have been generics and assertions and neither of those require BM changes.

There's a number of them, one of which I'm particularly hot on called immutable objects: a way to declare an object immutable. It means to declare that a class is final with all final fields, but with a twist – the quality operator is somewhat different. The motivation is that a sufficiently good optimizing JIT can optimize away the existence of these objects. So if you did something, such as complex numbers in terms of immutable objects, you could end up with code that did complex numbers as efficiently as FORTRAN.

> "There's a way to do a UI toolkit that's supported by IDE tools, yet projects its UI across the Web. That sounds to me like a pretty major announcement, but it's not us doing it"

Right now the big blockage in getting C++ or Java performance up to FORTRAN in a number of these numerical issues is that there's a limit as to how far you can optimize the primitive objects. In particular, the major block is that they still have an identity as an object. If you can get to where the notion of identity disappears so EQ goes away, you have only equal, and the things can mutate, then you can do copies arbitrarily, which is the number one thing you can do to allocate objects into registers where they aren't really objects, just things that live there momentarily. The things that you can do are pretty terrific.

<wyman>: **Like a new pseudo-primitive.**

<gosling>: Yeah, although there have been about a dozen sketches like that done, it's actually possible to properly do it so it actually doesn't affect the Java VM specification at all. I mean that in the sense of correctness, namely a correct VM would just ignore it, but then these immutable objects would be allocated like regular. However, if you really wanted to take advantage of this, there's a huge amount of work in the optimizer. Then you could start doing things like 3D graphics at unbelievable performance because with things like point beta structures, you could do some very amusing optimizations with them.

<wyman>: **As a big Mandelbrot set fan, I would love to see a complex as a pseudo-primitive.**

<gosling>: You can certainly do complex numbers now, the issue being that you can't get to FORTRAN optimization level with it. One answer to it is to make complex be a primitive the way FORTRAN does, but to do something that has the same optimization opportunities, that's general enough, so you can define complex for use in your Mandelbrot sets. I can define 3D points for doing constructive solid geometry and I get exactly the same kind of rocket optimization.

<wyman>: **Is the automatic optimization in JavaC and other compilers? Do you see the dynamic versus static issue anymore? There used to be a real issue doing static analysis of a program because you always had to worry about something slipping into the classpath at runtime. But static analysis is precluded by that dynamic nature, wouldn't you say?**

<gosling>: You can do static compilers and get the dynamics right, the problem is that it's difficult and the people who built static compilers were on the lazy side. The average static compiler was built by retreading someone's C compiler's back end and all kinds of funny things creep in. Dynamic compilers have gotten a lot better and they're pretty consistently beating the static compilers; since they know what your program is doing and what chip it's running on, they can do a lot more interesting things. They know what's loaded in, and it's been interesting watching the evolution of the programming style as it constructs large systems because more and more of what people do to construct systems is they don't build a big monolith; they build a spine that's a central place where things can plug in.

A Web server is a prime example, the way the Tomcat works with plug-in JSP pages and servlet pages. EE this and who knows what else. People roll in their own APIs all over the place. As a general technique for building flexible systems that can dynamically upgrade and all that kind of stuff, it's used all over the place. It's hard to find a Java program that doesn't use dynamic loading. Many people aren't even aware when it happens because with many of the underlying systems doing it, things like localization, they'll dynamically link in stuff, depending on where you happen to be. You don't have something where all the kanji you type in modules are now loaded for almost everybody; for the folks in Japan, it's right there, so you get what you need.

<williamson>: **Well, James, thank you for taking the time out to talk with us, and we look forward to seeing you again in September at JDJEdge.**

STRATEGIES FOR STORING JAVA OBJECTS IN RELATIONAL DATABASE

APPLY THESE FUNDAMENTALS FOR YEARS TO COME

WRITTEN BY SCOTT W. AMBLER

Like it or not, the majority of Java developers use a relational database (RDB), such as Oracle, Sybase, or MySQL, to persist their objects. That's reality for most people, so let's deal with it.

What's the big deal about storing Java objects in relational databases? Something called the object-relational impedance mismatch. The object paradigm is based on proven software engineering principles for building applications out of objects that have both data and behavior, whereas the relational paradigm is based on proven mathematical principles for efficiently storing data.

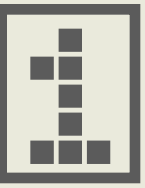
The impedance mismatch comes into play when you look at the preferred approach to access: with the object paradigm you traverse Java objects via their relationships whereas with the relational paradigm you duplicate data to join the rows in tables. This fundamental difference results in a nonideal combination of Java and RDBs, although when have you ever used two different technologies together without a few hitches? One of the secrets of success for mapping Java objects to relational databases is to understand both paradigms – and their differences – and then make intelligent tradeoffs based on that knowledge.

In this article, I present strategies for storing Java objects in RDBs. These strategies cover the following topics:

- Use persistent object IDs
- Map an attribute to zero or more columns
- Map a class to zero or more tables
- Map inheritance structures
- Use one data entity for an entire class hierarchy
- Use one data entity per concrete class
- Use one data entity per class
- Implement associations in relational databases
- Encapsulate persistence code

This article is modified from Chapters 7 and 8 of Ambler's latest book, *The Object Primer 2/e*, www.ambysoft.com/theObjectPrimer.html.

STRATEGY
Use Persistent Object IDs



The first thing you need to do is get a handle on the key strategy for your database, a likely source of contention between Java developers and database administrators because objects force you to rethink some of the precepts of data modeling's approach to assigning keys.

To store data in an RDB, you need to assign unique identifiers to the data rows representing your object to be able to identify them. In relational terminology, a unique identifier is called a *key*; in object terminology, it is called a *persistent object identifier* (POID) or simply an *object identifier* (OID). POIDs are often implemented as full-fledged objects in your Java applications, for example, the EJB specification enforces this concept in its primary key class for entity beans, and as large integers or strings in your database tables.

A critical issue that needs to be pointed out is that POIDs should have absolutely no business meaning whatsoever, which goes against common data modeling practices. Any table column with a business meaning can potentially change. And what's the one thing we learned as an industry over the years in the relational world?

It's a fatal mistake to give your keys meaning. No doubt about it.

If your users decide to change the business meaning (perhaps they want to add some digits or make a number alphanumeric), you need to make changes to your database in every single spot where you use that information. Anything that is used as a primary key in one table is likely used in other tables as foreign keys. What should be a simple change (adding a digit to your customer number), can be a huge maintenance nightmare. Yuck. In the relational database world, keys without business meaning are called surrogate keys.

POIDs allow you to simplify your key strategy within a relational database. Although POIDs do not completely solve the navigation/traversal issue between objects, they do make it easier. You still need to perform table joins (assuming you don't intend to traverse, to read in an aggregate of objects such as an invoice and all of its line items), but at least it's doable.

Another advantage is that the use of POIDs enables you to easily automate the maintenance of relationships between objects. When all of your tables are keyed on the same type of column(s), in this case POIDs, it becomes straightforward to write generic code to take advantage of this fact.

A POID should be unique within a class hierarchy, and ideally unique among all objects generated by your organization (something often called *global uniqueness*). For example, will the POID for a customer object be unique only for instances of customer, to people in general, or to all objects? Given the POID value 74656, will it be assigned to a student object, a professor object, and a seminar object? Will it be assigned to a student but not a professor (because the Student class and the Professor class are in the same class hierarchy)? Or will it only be assigned to a student object and that's it?

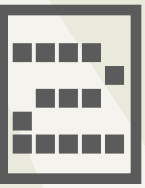
The real issue is one of polymorphism: it is probable that a student object may one day become a professor object, but likely not a seminar object. To avoid this issue of reassigning POIDs when an object changes type, you at least want uniqueness at the class hierarchy level, although uniqueness across all classes completely avoids this issue. This can be another point of contention between Java developers and data professionals – polymorphism is an important concept in the object world but not the data world.

There are several ways that you can generate values for POIDS:

1. Use the MAX() function (and add 1) on the POID column.
2. Maintain a separate table for storing the next value of a key.
3. Use Universally Unique Identifiers (UUIDs) from the Open Software Foundation.
4. Use Globally Unique Identifiers (GUIDs) from Microsoft.
5. Use proprietary database essential generation functions.
6. Use the HIGH/LOW approach. See the article "Enterprise Ready Object IDs" (www.sdmagazine.com/articles/1999/9912/9912p/9912p.htm)

STRATEGY

Map an Attribute to Zero or More Columns

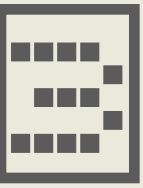


An attribute of a class will map to zero or more columns in a relational database. It's important to remember that not all attributes are persistent. For example, an Invoice Java object may have a grandTotal attribute that is used by its instances for calculation purposes but is not saved to the database. To complicate matters, some attributes of an object are objects in their own right, which in turn need to be mapped to your database. For example, a course object has an instance of TextBook as an attribute, which maps to several columns in the database.

The important thing is that this is a recursive definition. At some point, the attribute will be mapped to zero or more columns. It is also possible that several attributes could map to one single column in a table. For example, a class representing an American ZIP code may have three numeric attributes, one representing each of the sections in a full ZIP code, whereas the ZIP code may be stored as a single column in an address table.

STRATEGY

Map a Class to Zero or More Tables



Classes map to tables, although often not directly. Except for simple databases, you will never have a one-to-one mapping of classes to tables. In the following sections, I discuss three strategies for implementing inheritance structures to a relational database and an example where dissimilar classes map to one table.

STRATEGY

Map Inheritance Structures

The concept of inheritance throws in several interesting twists when saving Java objects into an RDB. There are three fundamental solutions for mapping inheritance into a relational database, and to understand them I discuss the trade-offs of mapping the class diagram presented in Figure 1. To keep the issues simple I have not modeled all of the attributes of the classes, nor have I modeled their full signatures, nor any of the methods of the classes.

FIGURE 1 Simple class hierarchy

FIGURE 2 Persistence model

Use One Data Entity for an Entire Class Hierarchy

With this approach you map an entire class hierarchy into one data entity, where all the attributes of all the classes in the hierarchy are stored. Figure 2 depicts the persistence model for the class hierarchy of Figure 1 when this approach is taken. Notice how a personPOID column was introduced for the primary key of the table – I will use POIDs in all of the solutions to be consistent and to take the best approach that I know of for assigning keys to data entities.

The advantages of this approach are that it is simple, that polymorphism is supported when a person changes roles, and that ad-hoc reporting is made easy because all of the data you

need about a person is found in one table.

The disadvantages are that every time a new attribute is added anywhere in the class hierarchy a new attribute needs to be added to the table. This increases the coupling within the class hierarchy. If a mistake is made when adding a single attribute, it could affect all the classes within the hierarchy and not just the subclasses of whatever class got the new attribute. It also potentially wastes a lot of space in the database and, in the case of some databases, forces you to rebind your code (by recompiling it) to the new data schema.

I also needed to add the objectType column to indicate if the row represents a student, a professor, or another type of person. This works well when someone has a single role but quickly breaks down if he or she has multiple roles (i.e., the person is both a student and an professor).

Use One Data Entity per Concrete Class

With this approach, each data entity includes both the attributes and the inherited attributes of the class that it represents. Figure 3 depicts the persistence model for the class hierarchy of Figure 1 when this approach is taken. There are data entities corresponding to each of the Student and the Professor classes because they are concrete, but not Person because it's abstract (indicated by the fact that its name is depicted in italics). Each of the data entities was assigned a primary key, studentPOID and professorPOID, respectively.

The main advantage of this approach is that it is still fairly easy to do ad-hoc reporting because all of the data you need about a single class is stored in only one table. There are several disadvantages, however.

First, when you modify a class, you need to modify its table and the table of any of its subclasses. For example, if you were to add height and weight to the Person class, you would need to add it to both tables – a lot of work.

FIGURE 3 Mapping each concrete class to a single data entity

Second, whenever an object changes its role, perhaps you hire one of our graduating students to become a professor, you need to copy the data into the appropriate table and assign it a new POID – once again a lot of work.

Third, it's difficult to support multiple roles and still maintain data integrity. For example, where would you store the name of someone who is both a student and a professor?

Use One Data Entity per Class

With this approach, you create one table per class, the attributes of which are the POID and the attributes that are specific to that class. Figure 4 depicts the persistence model for the data model for Figure 1 when this approach is taken. Notice how personPOID is used as the primary key for all three data entities. An interesting feature of Figure 4 is that the personPOID column in both Professor and Student is assigned two tagged values, one indicating that it forms both the primary and foreign keys for those tables.

The main advantage of this approach is that it conforms well to object-oriented concepts. It supports polymorphism because you have records in the appropriate tables for each role that an object might have. It's also easy to modify superclasses and add new subclasses as you merely need to modify/add one table.

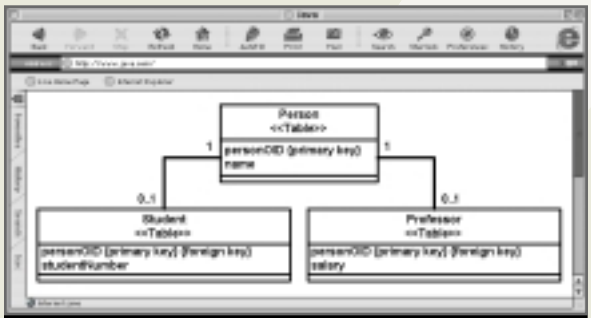


FIGURE 4 Mapping each class to its own data entity

There are several disadvantages to this approach. First, there are many tables in the database, one for every class (plus tables to maintain relationships). Second, it takes longer to read and write data using this technique because you need to access multiple tables. This problem can be alleviated if you organize your database intelligently by putting each table within a class hierarchy on different physical drive volumes. Third, ad-hoc reporting on your database is difficult, unless you add views to simulate the desired tables.

STRATEGY

Implement Associations in Relational Databases

Relationships in relational databases are maintained through the use of foreign keys. A foreign key is a data attribute(s) that appears in one table that may be part of – or is coincidental – with the key of another table. Foreign keys allow you to relate a row in one table with a row in another. To implement one-to-one and one-to-many relationships you merely have to include the key of one table in the other table.

In Figure 5 you see four tables, their keys (POIDs of course), and the foreign keys used to implement the relationships between them. First, there is a one-to-one association between the Position and Employee data entities. A one-to-one association is one in which the maximums of each of its multiplicities are one.

To implement this relationship I used the attribute positionPOID, the key of the Position data entity, in the Employee data entity. I was forced to do it this way because

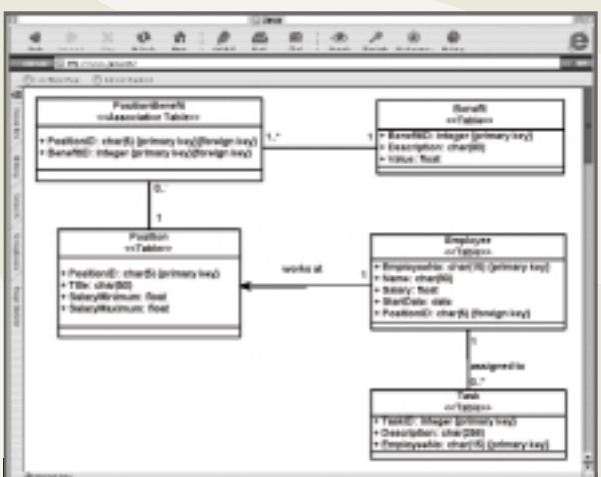


FIGURE 5 A persistence model for a simple human resources database

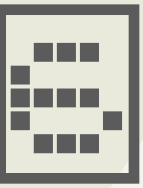
the association is unidirectional – employee rows know about their position rows but not the other way around. Had this been a bidirectional association, I would have needed to add a foreign key called employeePOID in Position as well.

Second, I implemented the many-to-one association (also referred to as a one-to-many association) between Employee and Task using the same sort of approach, the only difference being that I had to put the foreign key in Task because it was on the many side of the relationship.

The PositionBenefit table is interesting because it is an associative table, one that implements a many-to-many association between the Position and Benefit tables. Unlike Java, relational databases cannot natively implement many-to-many associations. Instead they need to be resolved with an associative table that implements the many-to-many as two one-to-many associations.

STRATEGY

Encapsulate Persistence Code



There are several ways you implement the persistence code within your software. The most common, and least palatable in my opinion, is to embed SQL statements in your classes. Listing 1 shows an example of how to persist the Order class; similar code would be needed to persist an instance of OrderItem, code that frankly isn't bulletproof. (I'll leave that as an exercise for you, please don't e-mail me.)

The advantage of this approach is that it allows you to write code quickly and is a viable approach for small applications and/or prototypes. There are two main disadvantages. First, it directly couples your business classes with the schema of your relational database, implying that a simple change (such as renaming a column or porting to another database) results in a rework of your source code. Second, it forces you to write significant amounts of SQL code, at least four statements for simple CRUD (create, retrieve, update, and delete) operations let alone the additional code required to support finding multiple instances based on defined criteria.

Castor	http://castor.exolab.org
CocoBase, Thought Inc.	www.thoughtinc.com
JdbStore, LPC	www.ilap.com/lpc/html/jdbstore.html
ObjectSpark	www.objectspark.com
Osage	http://osage.sourceforge.net
Toplink, WebGain Inc.	www.webgain.com/products/toplink/

TABLE 1 Java persistence frameworks

A slightly better approach is when the SQL statements for your business classes are encapsulated in one or more data classes. Listing 2 shows how this code would be invoked from the Order class (the code for the OrderData class would be similar to that in Listing 1). As you can see, there is significantly less code in the business objects, although you would still have the data classes to develop and maintain. Once again, this approach is suitable for prototypes and small systems of less than 40 to 50 business classes, but it still results in a recompilation (of your data classes) when simple changes to the database are made.

Your data classes are typically implemented as normal Java classes with embedded SQL, one data class for each business class (e.g., Employee has a corresponding EmployeeData class) or as a collection of stored procedures in your database (one or more for retrieving, one or more for deleting, and so on). The best thing that can be said about this approach is that you have at least encapsulated the source code that handles the hard-coded interactions in one place – the data classes.

A third approach is to use a persistence framework (Table 1 lists several for Java) that persist objects in RDBs in such a manner that simple changes to the relational schema often do not affect your object-oriented code. They do this by storing your mapping information in metadata, and then using that metadata to generate the source code needed to persist your Java objects. When your object or data schema changes you merely need to update your metadata instead of your source code.

The advantage of this approach is that only the person maintaining the metadata needs to understand the two schemas; your application programmers don't and, in fact, they don't even need to know that their objects are being stored in an RDB.

The main disadvantage is the misconceptions regarding performance that many Java developers have with respect to persistence frameworks, assuming they can achieve the best performance in their code by hardcoding SQL statements into it. This may be true if they happen to be experts at writing high-performance database access code, but this is seldom the case.

Yes, in theory, persistence frameworks add a bit of overhead to your Java code when compared to exceptionally well-written code. However, the reality is that the people who build persistence frameworks specialize in high-performance database access – they know a lot of tricks that you don't. My suggestion is to approach persistence frameworks with an open mind and to try them out. When you do so, I suspect you'll be pleasantly surprised at how well they actually work – as well as at the significant time and cost savings that they provide.

Java and Relational Databases Are Here to Stay

Storing your objects in relational databases is a reality for the vast majority of Java developers. In this article I presented some basic techniques for doing so, scratching the surface of this complex topic. The strategies I presented have been proven in practice, in fact they are fundamental strategies that you'll be able to apply for years to come regardless of the language you are working with.

I've personally used them on C++, Smalltalk, and Java projects. If you are interested in this topic, I highly suggest reading my new book *The Object Primer 2/e*, in which I show how to build business applications from end to end. This is done

RELATIONAL TERMINOLOGY

Column: The relational database equivalent of an attribute of a data entity stored in a relational table.

Row: The relational database equivalent of an instance of a data entity stored in a relational table. Also called a record or tuple.

Key: One or more columns in a relational data table that when combined form a unique identifier for each record in the table.

Composite key: A key consisting of two or more columns.

Surrogate key: A key without a business meaning.

Persistent object identifier (POID): A unique identifier assigned to objects, typically a large integer number. POIDs are the object-oriented equivalent of keys in the relational world.

SQL statement: A piece of SQL code used to retrieve, update, insert, or delete data or manipulate the schema of a relational database.

SQL: Structured Query Language.

from the point of view of a developer, starting with requirements and then moving through analysis to design and finally into implementation using Java and relational databases on the back end.

Recommended Reading

1. Ambler, S.W. *Building Object Applications That Work: Your Step-by-Step* (1998). *Handbook for Developing Robust Systems with Object Technology.* SIGS Books/Cambridge University Press. www.ambysoft.com/buildingObjectApplications.html
2. Ambler, S.W. "Enterprise Ready Object IDs." (1999). www.sdmagazine.com/articles/1999/9912/9912p/9912p.htm
3. Ambler, S.W. (2001). *The Object Primer 2nd Edition: The Application Developer's Guide to Object Orientation.* New York: Cambridge University Press. www.ambysoft.com/theObjectPrimer.html.
4. Szyperski C. *C* (1998). *Component Software: Beyond Object-Oriented Programming.* New York: ACM Press.
5. UUID Class Javadoc. http://jigsaw.w3.org/Doc/Programmer/api/org/w3c/util/UUID.html
6. Vermeulen, A., Ambler, S.W., Bumgardner, G., Metz, E., Misfeldt, T., Shur, J., & Thompson, P. (2000). *The Elements of Java Style.* New York: Cambridge University Press. www.ambysoft.com/elementsJavaStyle.html

About the Author

Scott W. Ambler is the president of Ronin International (www.ronin-intl.com) and thought leader of the Agile Modeling (AM) methodology (www.agilemodeling.com). He is the author of The Object Primer 2/e and co-author of The Elements of Java Style, both from Cambridge University Press. He is also author of the forthcoming Agile Modeling and co-author of the forthcoming Mastering EJB 2/e, both from John Wiley & Sons. In his spare time, Scott studies Goju karate, Kobudo karate, and Tai Chi.

scott.ambler@ronin-intl.com

Listing 1: Embedded SQL Code in the Order Class

```
    /**
     * Save an order and its aggregate order Items
     */
    private void save(Connection connection) throws SQLException {

        PreparedStatement orderStatement = null;
        PreparedStatement orderItemStatement = null;
        Vector items;

        //  Build statements to either insert or update
        if (isPersistent() ) {
            orderStatement = connection.prepareStatement(
"Update INTO Order VALUES(?,?,?,?,?,?,?)");
            orderItemStatement =
connection.prepareStatement(OrderItem.getUpdateSQL());
        } else {
            orderStatement = connection.prepareStatement(
"INSERT INTO Order VALUES(?,?,?,?,?,?,?)");
            orderItemStatement =
connection.prepareStatement(OrderItem.getInsertSQL());
        }

   //  Add the order information from this object
        orderStatement.setInt(1, getOrderID());
        orderStatement.setInt(2, getCustomerNumber());
        orderStatement.setDate(3, getOrderDate());
        orderStatement.setDouble(4, getFederalTax().getNumber());
        orderStatement.setDouble(5, getStateTax().getNumber());
        orderStatement.setDouble(6, getLocalTax().getNumber());
        orderStatement.setDouble(7, getSubtotal().getNumber());

        //  Save the order and order contact information
        orderStatement.executeUpdate();
        orderStatement.close();

        //  Save the order items
        items = getOrderItems();
        for ( int i = 1; i <= items.size(); i++ ) {
            OrderItem item = (OrderItem) items.elementAt(i);
            item.save(orderItemStatement, getOrderID(), i);
        }
        orderItemStatement.close();

    }
    /**
     * Refreshes this object with the data for
     * the order with the given id.
     */
    public void retrieve(Connection connection)
        throws SQLException
    {
        OrderContact ship, bill;
        int shipToID, billToID;

        // Retrieve the record from the Order table

        PreparedStatement statement =
        connection.prepareStatement("SELECT * FROM Order WHERE
OrderID = ?");
        statement.setInt(getOrderID());
        ResultSet rs = statement.executeUpdate();
        rs.next();

        setCustomerNumber(rs.getInt(2));
        setOrderDate(rs.getDate(3));
        setFederalTax(rs.getDouble(4));
        setStateTax(rs.getDouble(5));
        setLocalTax(rs.getDouble(6));
        setSubtotal(rs.getDouble(7));
        setIsPersistent(true);

        // Read the records from the OrderItem table
        try {
            Vector items = OrderItem.retrieveOrderItems(orderID);
            setOrderItems(items);
        } catch ( Exception ex ) {
            //  Handle the exception appropriately...
        }
    }

    /**
     * Delete the order and Its order Items
     */
    private void delete(Connection connection)
    throws SQLException {

        PreparedStatement ps = null;
        Vector items;

   ps = connection.prepareStatement( "DELETE Order WHERE OrderID =
?");

        ps.setInt(1, getOrderID());
        ps.executeUpdate();
        ps.close();

        //  Add deletion of the order items to the transaction
        items = getOrderItems();

        for ( int i = 1; i <= items.size(); i++ ) {
            OrderItem item = (OrderItem) items.elementAt(i);
            ps = item.delete(connection);
            ps.executeUpdate();
        }
    }
```

Listing 2: Delegating Persistence of Order to OrderData

```
    /**
     * Save an order and its aggregate order Items
     */
    private void save(Connection connection) throws SQLException {
        OrderData.save(this, connection);
    }

    /**
     * Refreshes this object with the data for
     * the order with the given id.
     */
    public void retrieve(Connection connection)
        throws SQLException
    {
        OrderData.retrieve(this, connection);
    }

    /**
     * Delete the order Its order Items
     */
    private void delete(Connection connection)
    throws SQLException {
        OrderData.delete(this, connection);
    }
```

Download the Code! www.JavaDevelopersJournal.com

JASON BRIGGS J2ME EDITOR

Out of Nowhere

If you've ever spent time in the Middle East, you'll know that bargaining is a way of life. You haggle over everything, especially if you're a tourist – they automatically triple and quadruple the price if you're a foreigner. So it doesn't seem that unusual to be arguing over the price of a bus fare to the center of Turkey – a reduction, when converted to sterling, works out to around 50 pence. So, in the flush of pride at finally having outwitted the locals and arguing the price down to an "acceptable" amount (which is still probably 2 quid more than the locals pay), you should not, on the face of it, be that surprised when the coach pulls up on the outskirts of some unknown city, depositing you, your backpack, and some equally bewildered Turks unceremoniously on the side of the road at 4 in the morning.

Looking back, I've come up with the perfect J2ME application for situations such as those – an instant travel guide: part phrase book, part translation guide, and part GPS-based map system. Ignore for a moment the small difficulties such as the fact that very few phones and even fewer PDAs (if any) have GPS facilities built in, and it does seem like an incredibly useful tool. At the time I'm sure I would've been quite happy to pay an extortionate U.S.$20 membership fee and three months' subscription in advance, just for the peace of mind of knowing exactly where on earth I was and a few useful phrases to converse in with my fellow travelers, also stranded on the roadside. Again, even before starting to travel, I might have been interested in a more reasonably priced subscription for three or four months, or perhaps a year.

I'm sure someone out there is going to argue that WAP is a likely enough candidate for this application; however, I'm equally sure that a Java version is a better idea. Designed well, it would present a slicker interface – relatively consistent, no matter what the platform. With a slightly heavier interface, the map rendering could be done on the client side, reducing the probable network traffic required.

What this example hopefully illustrates is that there are real-world applications in which J2ME just fits. Part 2 of the beginner's guide to MIDP, found in this issue, will demonstrate one such application, simplified greatly by the use of Java. You'll also find a review of one of the ever-growing number of devices that could run this sort of app: the Compaq iPAQ. For good luck, we also review one of an also-increasing number of J2ME virtual machines: Insignia's Jeode.

As expected, there were a number of important announcements at JavaOne covering a bewildering range of products and APIs. In some ways I'm glad I wasn't there, so Alan (our E-I-C) has to summarize it all for me. That's what I keep telling myself in those moments when my skin turns that delicate shade of green. Must be something to do with my diet.

A couple of the more surprising pieces of news: Sony announced the integration of Java into the PlayStation 2 console, and Sun proclaimed a new J2ME Games Profile. Java is definitely underrepresented in the console market with only Sega's ill-fated Dreamcast supporting a version of PersonalJava so far. At this point it seems likely that one will have an impact on the other, since Sony is involved in the Games Profile specification, but what it will mean for you and me, as developers, is not yet clear. Stay tuned.

By the way, while standing on the side of a Turkish road in the middle of the night, without a clue what to do, a cab will generally come hurtling out of nowhere, the driver pretending he hasn't seen you, and still somehow manage to stop right next to you by applying the brakes at the last possible instant. He will, of course, also speak English. It's one of those unfathomable, natural laws.

jasonbriggs@sys-con.com

AUTHOR BIO

Jason Briggs works as a Java analyst programmer in London. He's been officially developing in Java for three-and-a-half years – unofficially for over four.

J2ME INDEX

J2ME FAQ

IS PERSONALJAVA PART OF J2ME?

The short answer is yes. For the long answer, we'll refer to Sun's FAQ for J2ME, which states that PersonalJava was the "first Micro Edition technology." Because PersonalJava has been around for a while now, you'll find more products with a version of it installed. But sometime this year (2001), Sun is expected to replace the existing PersonalJava technology – based on Java 1.1 – with a new release based upon Java 2, and incorporated into the J2ME concepts of Configuration and Profile components.

IS ALL THE JAVA API WITHIN J2ME?

No. Even PersonalJava – which has the most complete coverage of the Standard Edition API – is still just a subset.

WHAT IS A "MIDLET"?

Actually, the correct word is MIDlet. A MIDlet is an application written for MIDP (the Mobile Information Device Profile). You might find these on mobile phones, PDAs – in general, small devices.

CAN I USE THREADS? IS THERE A PENALTY?

Yes, you can use threads, unless you're writing a JavaCard applet. As for the penalties, it very much depends upon how you want to use them, and the environment you are working within. When developing for constrained devices, you always have to keep the resources you have available in the back of your mind. If you're writing a MIDlet, and create 100 threads to try to load 100 images simultaneously, then there definitely will be a penalty – it undoubtedly won't work.

DO I USE AWT OR SWING FOR MY GUI?

If you're developing a PersonalJava application, then you have access to a modified version of AWT – "modified" meaning that a few java.awt classes/methods are optional, that some have been changed, and that there are some additions to the basic package. You may be able to get Swing to work within a PersonalJava environment as well. A brief skim of the PersonalJava forums show some success stories – and more than a few painful attempts. None of the other J2ME "products" support AWT or Swing (for example, MIDP has the javax.microedition.lcdui package, for user interfaces).

WHERE CAN I FIND MORE INFORMATION ABOUT WIRELESS TECHNOLOGIES?

The back issues of *JDJ* are one place you can look. For online information, you can look at the following URLs:

1. http://developer.java.sun.com/developer/products/wireless/
2. *Bill Day's J2ME archive:* www.billday.com/j2me/
3. *Sun's Wireless forums:* http://forum.java.sun.com/

WHERE CAN I DOWNLOAD J2ME EMULATORS?

The J2ME Wireless Toolkit: http://java.sun.com/products/j2mewtoolkit/download.html

To download the MIDP reference implementation on this page: http://java.sun.com/products/midp/

CLDC : www.sun.com/software/communitysource/j2me/cldc/download.html

CDC (and the Foundation profile): www.sun.com/software/communitysource/j2me/cdc/download.html

WHERE CAN I FIND DEVICES THAT RUN J2ME?

Move to another country. At the moment, there are a limited number of countries where J2ME capable devices have been released – especially for mobile phones. While you can probably find PDAs that support PersonalJava almost anywhere in the world, the same is not true for mobiles.

In Japan, NTT DoCoMo has a number of phones from Panasonic, Fujitsu, Sony, and others (only available in Japan, of course). In the U.S., Motorola has a couple of J2ME capable mobiles. For a more comprehensive list, check out www.javamobiles.com/

J2EE Design Patterns: The Next Frontier

–continued from page 10

bimonthly column based on his popular book *Core J2EE Patterns*. There is a J2EE Patterns community out there. Feel free to write to us to find out how you can learn more about J2EE Patterns. Among other excellent articles, we also have a piece on J2EE frameworks by Steven Randolf that should help you design better applications by reusing J2EE framework components. We also bring you reviews on a couple of books that address J2EE design and real-world application of J2EE frameworks.

So, return to the J2EE section, and as the Roman emissary in my Asterix comics would say: "make less factories, build much objects."

Java Comes of Age

–continued from page 44

father of Java himself, James Gosling, and Dr. Alan Baratz, formerly president of JavaSoft and of the Software Products & Platforms division at Sun Microsystems, now CEO of Zaplet, Inc.

As well as a full program of richly varied technical sessions on J2SE, J2EE, and J2ME, there will also be a high-powered SuperSession in which key industry figures will dissect for delegates and attendees the fast-emerging new Web Services Paradigm of distributed Internet applications.

It's all at the Hilton, New York, on September 23–26. I'll see you there!

WRITTEN BY ROB MACAULAY

Fiat Lux

Personally, I think J2ME is what Java is really about. Let's leave aside the fact that Java was originally developed (as project Oak) for just this purpose, and see what it means today.

The demand for embedded microprocessor systems has exploded, as consumers and equipment manufacturers require systems that incorporate more "intelligence." The "intelligence" is used in a number of different ways. Systems can, of course, be used to automate complex tasks without user intervention – the traditional requirement. However, there's also a trend toward systems that are adaptive to changing needs and can communicate with other equipment or with other applications running on the same device.

These extra requirements are putting traditional methods of designing embedded systems under increasing strain. Conventional embedded systems can suffer from the following problems: increased complexity and device count, lack of portability of solutions, lack of support for networking, poor performance, and so forth. Designing a working embedded system has been something of a black art, and the number of engineers with the required skills are in short supply.

Java, and J2ME in particular, offers a number of solutions to these problems, and also offers the promise of completely new ways of building embedded systems.

The most obviously helpful feature of Java is the notion of "write once, run anywhere." This is achieved through the distribution of code via class files, and also through the definition of the APIs. In the past, it was extremely difficult to port an application from one embedded platform to another. With Java it'll be easier to sell your application into multiple markets.

Of course, the big bugbear of using Java in embedded environments was always the large amount of resources required to run interpreted or Just-In-Time (JIT) JVMs – usually a desktop PC. Now that we have J2ME and, in particular, Java processors, Java in embedded systems will really take off.

However, my vision is a little different from what you might expect. When most of us start eulogizing about J2ME and Java processors, it's all about cheap color PDAs using Java. Me, I get excited about light switches. My company makes native processors for Java-based applications, and these are already small enough so we can see that the cost of a Java-based controller will, in five years time, be cheap enough to put in your light switch.

Why would you want to run Java on your light switch? If you need to control your lights, you need some way of communicating with them, and this brings in the other part of the equation: wireless. Cheap wireless technologies change the picture completely because they do two things: slash installation costs and allow you to work with any controller that supports wireless technology. However, technologies such as Bluetooth need Java; the communication protocols need a reasonable amount of processing power to manage them. In the past, providing this resource for a light switch was unthinkable, but now it's just around the corner.

Java immediately brings other benefits too. The light switch is now capable of announcing itself to any passing controller, which can discover what capabilities it has and could even download a controller applet from the switch.

And what is that passing controller? Very probably a mobile phone running – guess what? – Java! These devices are starting to appear in the shops right now, so this is not science fiction. Using mobile phones as controllers also means you get wide-area networking for free. Mind you, it might be disconcerting to find that your huge phone bill is due to the light switch talking to its friend in Japan.

Do I have to change all my light switches? No, of course not. The light switch simply downloads a new interface, or possibly the controller downloads an adapter class. Easy! And automatic too – the user wouldn't need to know it has happened.

The light switch is perhaps a facetious example; it certainly won't be the first application of the technology, and it sounds a bit like overkill. However, I think it's the ultimate point of this sort of technology: cheap, controllable, and above all, adaptable.

rob@vulcanmachines.com

AUTHOR BIO

Rob is a founder and technical director of Vulcan Machines, designers of native processors for Java-based applications. Rob has been involved in the manufacture of semiconductors for nearly 20 years and has been designing microprocessors for 10. In the last three years, he has concentrated exclusively on Java technology.

SYS-CON MEDIA

PUBLISHER, PRESIDENT, AND CEO
FUAT A. KIRCAALI fuat@sys-con.com

ADVERTISING
SENIOR VICE PRESIDENT, SALES AND MARKETING
CARMEN GONZALEZ carmen@sys-con.com
VICE PRESIDENT, SALES AND MARKETING
MILES SILVERMAN miles@sys-con.com
ADVERTISING SALES DIRECTOR
ROBYN FORMA robyn@sys-con.com
ADVERTISING ACCOUNT MANAGER
MEGAN RING megan@sys-con.com
ASSOCIATE SALES MANAGER
CARRIE GEBERT carrieg@sys-con.com
ASSOCIATE SALES MANAGER
CHRISTINE RUSSELL christine@sys-con.com
SALES ASSISTANT
ALISA CATALANO alisa@sys-con.com

EDITORIAL
EXECUTIVE EDITOR
M'LOU PINKHAM mpinkham@sys-con.com
EDITOR
NANCY VALENTINE nancy@sys-con.com
MANAGING EDITOR
CHERYL VAN SISE cheryl@sys-con.com
ASSOCIATE EDITOR
JAMIE MATUSOW jamie@sys-con.com
ASSOCIATE EDITOR
GAIL SCHULTZ gail@sys-con.com
ASSOCIATE EDITOR
BRENDA BREENE brenda@sys-con.com
ASSISTANT EDITOR
LIN GOETZ lin@sys-con.com

PRODUCTION
VICE PRESIDENT, PRODUCTION AND DESIGN
JIM MORGAN jim@sys-con.com
ART DIRECTOR
ALEX BOTERO alex@sys-con.com
ASSOCIATE ART DIRECTOR
LOUIS F. CUFFARI louis@sys-con.com
ASSISTANT ART DIRECTOR
CATHRYN BURAK cathyb@sys-con.com
GRAPHIC DESIGNER
ABRAHAM ADDO abraham@sys-con.com
GRAPHIC DESIGNER
RICHARD SILVERBERG richards@sys-con.com
GRAPHIC DESIGNER
AARATHI VENKATARAMAN aarathi@sys-con.com

WEB SERVICES
WEBMASTER
ROBERT DIAMOND robert@sys-con.com
WEB DESIGNER
STEPHEN KILMURRAY stephen@sys-con.com
WEB DESIGNER
PURVA DAVE purva@sys-con.com
WEB DESIGNER INTERN
CAROL AUSLANDER carol@sys-con.com

ACCOUNTING
ASSISTANT CONTROLLER
JUDITH CALNAN judith@sys-con.com
ACCOUNTS PAYABLE
JOAN LAROSE joan@sys-con.com

SYS-CON EVENTS
VICE PRESIDENT, SYS-CON EVENTS
CATHY WALTERS cathyw@sys-con.com
CONFERENCE DIRECTOR
DANIELLE NAPPI danielle@sys-con.com
CONFERENCE MANAGER
MICHAEL LYNCH mike@sys-con.com
SALES EXECUTIVE, EXHIBITS
MICHAEL PESICK michael@sys-con.com
SALES EXECUTIVE, EXHIBITS
RICHARD ANDERSON richard@sys-con.com
SHOW ASSISTANT
NIKI PANAGOPOULOS niki@sys-con.com
JDJSTORE.COM
ANTHONY D. SPITZER tony@sys-con.com

The Great J2ME API Rundown

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

CONNECTED, LIMITED DEVICE CONFIGURATION (CLDC) – VERSION 1.0

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

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

CONNECTED DEVICE CONFIGURATION (CDC) – VERSION 0.2

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

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

MOBILE INFORMATION DEVICE PROFILE – VERSION 1.0

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

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

FOUNDATION PROFILE – VERSION 0.2

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

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

J2ME GAME PROFILE

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

1. 3D Modeling and Rendering for Games
2. 3D Physics Modeling for Games
3. 3D Character Animation for Games
4. 2D Rendering and Video Buffer Flipping for Games
5. Game Marshalling and Networked Communication
6. Streaming Media for Games
7. Sound for Games
8. Game Controllers
9. Hardware Access for Games

Stayed tuned to *JDJ* – we'll attempt to bring more information as it comes to hand.

PERSONALJAVA SPECIFICATION – VERSION 1.2A

java.applet	full support from JDK1.1.8
java.awt	modified from JDK1.1.8 • note: there is an extra method for PJ for double-buffering in java.awt.Component
java.awt.datatransfer	full support
java.awt.event	full support
java.awt.image	full support
java.awt.peer	modified
java.beans	full support
java.io	modified
java.lang	modified
java.lang.reflect	modified
java.math	optional – may or may not be supported
java.net	modified
java.rmi	optional
java.rmi.dgc	optional
java.rmi.registry	optional
java.rmi.server	optional
java.security	modified
java.security.acl	unsupported
java.security.cert	some classes required, some optional
java.security.interfaces	required if code signing is included
java.security.spec	required if code signing is included
java.sql	optional
java.text	full support
java.text.resources	modified
java.util	modified
java.util.jar	required if code signing is included
java.util.zip	modified

Additional PersonalJava specific packages are:

com.sun.awt	for mouseless environments
com.sun.lang	a couple of error & exception classes
com.sun.util	for handling timer events

PersonalJava will eventually be superseded by the Personal Profile. For more information on the PersonalJava Application Environment: http://java.sun.com/products/personaljava/

JAVA TV – VERSION 1.0

javax.tv.carousel	access to broadcast file and directory data
javax.tv.graphics	root container access and alpha blending
javax.tv.locator	referencing data and resources
javax.tv.media	controls and events for man agement of real-time media
javax.tv.media.protocol	access to generic streaming data in a broadcast
javax.tv.net	IP datagram access
javax.tv.service	service information access
javax.tv.service.guide	supporting electronic program guides
javax.tv.service.navigation	services and hierarchical service information navigation
javax.tv.service.selection	select a service for presentation
javax.tv.service.transport	information about transport mechanisms
javax.tv.util	creating and managing timer events
javax.tv.xlet	communications interfaces used by apps and the app manager

Get off that couch and check out the Java TV page at the following URL: http://java.sun.com/products/javatv/

JAVA EMBEDDED SERVER – VERSION 2.0

com.sun.jes.service.http	servlet/resource registrations
com.sun.jes.service.http.auth.basic	http basic authentication
com.sun.jes.service.http.auth.users	management of users and their access
com.sun.jes.service.timer	for handling timer events
org.osgi.framework	consistent model for app. dev., supports dev. and use of services
org.osgi.service.device	detection of devices
org.osgi.service.http	http access of resources
org.osgi.service.log	logging facility

You can find more information on Embedded Server on the following site: www.sun.com/software/embeddedserver/

JAVA CARD – VERSION 2.1.1

java.lang	fundamental classes
javacard.framework	core functionality of a JC applet
javacard.security	security framework
javacardx.crypto	extension package with security classes and interfaces

Next time you use that American Express Blue card, you may want to know how it works, so take a look here: http://java.sun.com/products/javacard/

Jeode

by Insignia Solutions

REVIEWED BY ANTHONY SIMMONS
anthony.simmons@mailandnews..com

specs

Insignia Solutions Inc.
41300 Christy St.
Fremont, California 94538
Web: www.insignia.com
Phone: 800 848-7677
E-mail: jeode@insignia.com

Test Environment
P800 with Windows NT 4.0 SP4

Founded in the U.K. in 1986, Insignia started out developing technology that enabled non-Intel computers to run DOS and Windows applications. Twelve years later, after a shift in focus, the first beta versions of the Jeode platform and Jeode Embedded Virtual Machine emerged. According to Insignia's statistics, more than 35 million runtime units of Jeode technology have been contracted by OEMs, OS, and middleware suppliers.

Jeode includes class libraries for either PersonalJava or EmbeddedJava (depending on which implementation the device manufacturer chooses), and a tool suite that includes a configurator, monitor, and deployment tools. Jeode is available on a variety of operating systems – Windows CE 2.12 and 3.0, Windows NT4, VxWorks, Linux, ITRON, Nucleus, BSDi UNIX, and pSOS – and also supports a number of microprocessors: ARM, MIPS, x86, Hitachi SuperH-3, Hitachi SuperH-4, and PowerPC. Putting the OS and microprocessor support together, you're likely to have a myriad array of devices to choose from.

If you're reading ***JDJ*** from back to front, or have flipped to this page in the bookstore (then stop loitering and buy it, this isn't a library, you know?), you should read the iPAQ review first. ***JDJ***'s review used Jeode as the virtual machine upon which to run test applications; so, to find out how Jeode performs on an actual device, look there first. However, in this review we look at Jeode running on a desktop Windows NT machine and compare its performance against a number of other VMs.

How Does It Work?

Jeode uses a tool called the JeodeConfigurator to configure the virtual machine before running an application. This is useful if you want to develop in the Windows NT environment (for example), but your application will be running on a more limited platform (such as WinCE). The Configurator allows you to tune properties such as memory size, maximum dynamic memory size, system memory, Java memory, and stack size. You can switch dynamic compilation on or off, set the space used for dynamic compiling, turn debugging on or off, and so on. If your target device has a minimum of 8MB of RAM available, you can quite easily set your memory size to 8MB and tweak other options accordingly.

The Tests

As with the iPAQ review there are three main tests I'll be running Jeode through to give a very basic idea of how it performs. Test 1 just displays some of the AWT components on the screen ensuring that the VM isn't doing anything odd. Test 2 draws four triangles using the drawLine() graphics method and gives the time taken for the draw. Test 3 turns a byte array of pixels into an image before drawing that image to the screen. In addition, I've also decided to run the Sequential Benchmarks from Java Grande (www.java-grande.org), which gives stats on base Java functionality (additions, divisions, array assignments, casts, etc.).

The five VMs I'll be comparing are Sun JDK 1.1.8, Sun JDK 1.3, Sun PersonalJava Emulation Environment, Insignia's Jeode 1.7 (of course), and Microsoft VM 5.00.2752. (*Note:* If you're looking for proper benchmarking software or benchmarks for various virtual machines, see References at the end of the review.)

RESULTS OF TEST 1

The only perceivable difference between the virtual machines for test 1 is that the version of PersonalJava that I'm using (PJEE3.1) uses the truffle peer set, hence the components look different to standard AWT. Apart from that, each VM behaves the same as the others.

RESULTS OF TEST 2

The results of this test, which was run in a 640x480 window, are shown in Table 1.

RESULTS OF TEST 3

The results for this test, again run in a 640x480 window, are shown in Table 2.

Table 3 shows the basic arithmetic operations provided in Section 1 of the Java Grande benchmark.

Conclusion

On WinNT, Jeode appears to hold its own against the competition, pulling in better results than the PersonalJava Emulation

SUN JDK 1.1.8	SUN JDK 1.3	SUN PJEE3.1	INSIGNIA'S JEODE 1.7	MS VM
6ms	8ms	14ms	7ms	5ms

TABLE 1 Test 2 results

SUN JDK 1.1.8	SUN JDK 1.3	SUN PJEE3.1	INSIGNIA'S JEODE 1.7	MS VM
41.39fps	38.17fps	1.41fps	25.97	46.44fps

TABLE 2 Test 3 results

Environment (which, in its defense, is not supposed to be a production VM), and fairly similar results to a 1.1 VM. Running on the iPAQ (see iPAQ review) Jeode manages to beat the Sun VM on the pixel blit test, but not on the drawLine creation of triangles. These are not comprehensive tests, of course, so I invite you to draw your own conclusions, depending upon the nature of the applications you're developing.

The ability to tweak your environment to mimic a device, before you even have the hardware, must count as a selling point. And with Compaq and Insignia releasing a $19.95 version of the software – at the very least, providing a link (see References) for customers interested in using your Java software is a no-brainer.

References

1. *The Compaq/Insignia Jeode deal:* www.compaq.com/products/handhelds/java.html
2. *Java Grande:* www.javagrande.org
3. *Java Performance Report:* www.javalobby.org/fr/html/frm/javalobby/features/jpr/
4. *CaffeineMark:* www.pendragon-software.com/pendragon/cm3/

	SUN JDK 1.1.8	SUN JDK 1.3	SUN PJEE3.1	JEODE 1.7	MS VM
Add:Int (adds/s)	342,023,616.0	423,154,144.0	5,484,367.5	330,135,392.0	294,593,472.0
Add:Long (adds/s)	130,653,904.0	125,009,064.0	5,105,004.0	136,632,960.0	144,011,424.0
Add:Float (adds/s)	6,175,184.5	6,496,946.5	2,935,569.5	5,888,018.5	6,636,422.5
Add:Double (adds/s)	6,168,210.0	6,480,500.0	2,616,249.2	5,743,532.0	5,674,309.0
Mult:Int (mult/s)	156,794,064.0	162,097,456.0	5,333,680.5	146,958,176.0	106,320,568.0
Mult:Long (mult/s)	81,603,784.0	31,864,452.0	4,595,019.0	67,299,240.0	43,600,560.0
Mult:Float (mult/s)	6,916,582.0	6,730,753.5	2,955,267.0	6,600,064.5	6,713,103.5
Mult:Double (mult/s)	6,586,268.0	6,417,045.0	2,582,760.5	5,900,742.0	5,090,095.5
Div:Int (div/s)	1,145,233,660.0	18,094,870.0	4,026,740.0	17,885,486.0	17,535,184.0
Div:Long (div/s)	186,619,200.0	5,711,497.0	2,843,259.8	6,549,932.0	7,426,344.0
Div:Float (div/s)	6,316,601.0	5,985,241.5	2,906,407.5	6,054,246.0	6,197,140.5
Div:Double (div/s)	6,378,075.0	6,096,599.0	2,532,772.8	5,686,125.0	5,274,612.0

TABLE 3 Basic arithmetic operations

Wireless Apps Wanted: Developers Only Need Apply

A market for third-party J2ME applications is about to explode

WRITTEN BY
KIMBERLY MARTIN

With the growth of the wireless industry, and telecommunications providers realizing the potential of provisioning personalized mobile applications for customers, Java developers are positioned to capitalize on a tremendous market opportunity. The number of wireless communications devices installed worldwide has already exceeded the number of desktop PCs, and more explosive growth is anticipated.

While new technologies create unprecedented opportunities for telecommunications companies, their core competencies lie in the network itself, not the content delivered on it, and beg for the development of compelling consumer products and services. As in other industries, content will become king and Java developers will play a key role in providing fresh content in the form of applications.

The rapid evolution of the wireless marketplace means carriers need skilled support to deliver value-added services, particularly with the imminent arrival and adoption of GPRS and 3G. The variety of applications that consumers will want is seemingly endless. While no one particular killer app for mobile use is in sight, it's pretty clear that the killer app for a teenaged girl in Texas is significantly different from that for a middle-aged man in Helsinki, despite the fact that both demographics are wired. Mobile operators simply won't keep up with demand for the variety of applications that the consumer will want unless they look to third-party developers to generate the applications and content for their network.

Sun's J2ME Creates Universal Standard for Wireless Devices

Until recently, mobile devices represented a developer's worst nightmare because they typically have no standard operating system, microbrowser, or form factor, meaning all applications had to be written for specific devices.

To combat these limitations, Sun has taken a step toward creating a product that addresses the need for a universal standard for wireless devices. Java 2 Micro Edition (J2ME) is Sun's version of Java aimed at machines with limited hardware resources, such as PDAs, cell phones, and other consumer electronic and embedded devices. J2ME addresses the needs of devices with as little as 128KB of RAM and with processors a lot less powerful than those used on typical desktop and server machines.

Like the other Java technologies (J2SE and J2EE), the J2ME platform maintains these advantages:

- Built-in consistency across products to run anywhere, anytime, over any device
- Portability of the code
- Leveraging of same Java programming language
- Safe network delivery

In addition, applications written with J2ME technology are upwardly scalable to work with the J2SE and J2EE platforms, creating a platform that enables thin-client applications to work in a larger client/server environment.

Beyond these cross-technology advantages, J2ME extends the promise of Java by eliminating the problem of device proliferation while also providing additional benefits. Through caching on the device, it solves the problem of spotty network coverage. It also provides graphic capabilities for wireless devices, representing even more development opportunities. Applications written in Java are future-proofed to evolve with the hardware. Devices that support Java permit easy upgrades of applications on hardware so that consumers can easily change or customize their devices as needed.

J2ME Gaining Widespread Acceptance

Network operators, such as Cingular, NTT DoCoMo, Vodafone, Telefonica, and others, have publicly proclaimed their support for J2ME. Device manufacturers, such as Nokia, Motorola, Siemens, and Research in Motion (RIM), are building J2ME into their next-generation wireless handsets. Soon, the majority of mobile phones, including less expensive models, will be able to run applications written in Java. In fact, Nokia has predicted that in 2002 it will sell more than 50-million handsets with Java, and even more by the end of 2003 – populating the world with at least 100-million Java-equipped mobile phones. Furthermore, J2ME technologies also played a major role at JavaOne 2001, Sun's worldwide Java developer conference, signaling the mainstream acceptance of Java on mobile devices, smart cards, and in embedded solutions.

Based on this widespread support, the message to developers is loud and clear – real J2ME-based devices are on sale today; network operators are beginning to deploy the technologies to support downloadable Java code to devices, and ASPs and traditional ISVs are currently developing software based on J2ME. Tremendous opportunity abounds. Now that the technologies are ready and rapid adoption is underway, developers should be hustling to build and deploy real solutions for Java-powered devices.

Consumers will also benefit. Since Java is platform-independent, programs will become vendor-independent; for example, apps written for Motorola phones will be able to run on mobile phones from NEC without changes. By selecting a Java-enabled phone, consumers won't be limited to games and applications bundled with the firmware on their phone, but will be able to replace them with new ones – downloadable anytime.

Keys to Success

To encourage the development community to build such applications, carriers must manage their channel effectively. They must make it easy for the developer to roll out applications to the consumer and provide the billing and metering services for the developer, ensuring royalties are tracked. A market for third-party J2ME applications will flourish when carriers can effectively target applications to specific consumers and direct payments to developers based on usage.

To effectively meet these two conditions, ATG has developed a service-delivery solution that will provide server infrastructure support for Java technology–enabled wireless devices. Named the ATG Mobile Application Provisioner, it helps the developer as well as the consumer. First, it provides developers with access to wireless carriers' customers and billing services so that the developer community can leverage the carriers as a distribution channel for their applications. Second, it uses scenario-driven personalization so that carriers can design highly targeted offers to their customers.

To keep the consumer happy, network operators must ensure that consumers are offered the right application, at the right time, on the right device. Consumers must view these new applications and services as a benefit, not a blatant mechanism for generating revenue on the carrier's part. Furthermore, consumers want one bill, and developers want a simplified way to track and collect royalties. Successful carriers will extend their already robust billing and metering capabilities to seamlessly bill consumers for new applications and services whether created by the carrier or by a third-party developer.

By considering contextual information such as time of day, location, and past behavior, carriers can target the offering of new applications to individual consumers. For instance, a consumer who has purchased a map of the New York City subway system while on a trip to Manhattan, would probably view it as a benefit to be offered a map of London's Tube system the next time he or she lands at Heathrow. This hyperpersonalization ensures that the consumer is not presented with irrelevant offerings, while also specifically targeting the application to qualified buyers. For the developer, this means there's a higher likelihood of purchase and increased revenue potential.

Through advanced personalization capabilities, carriers that provide feedback to the developer on the application's success will ensure that their content is always fresh and always relevant. For example, a developer of a wildly successful 1.0 version of a game should be encouraged monetarily to develop a version 2.0. Likewise, targeting those users of version 1.0 should provide a great base of customers for the 2.0 version.

The convergence of wireless hardware and software is inevitable. J2ME is the current leader and standard by which this convergence is starting. By effectively managing relationships with suppliers and consumers in the development community, carriers will be able to reap the rewards of J2ME.

Author Bio

Kimberly Martin is a product manager for ATG, a provider of relationship management and e-commerce products and services. She's responsible for product direction and strategy for the mobile and wireless market.

kmartin@atg.com

iPAQ

by Compaq Computer Corporation

REVIEWED BY ANTHONY SIMMONS
anthony.simmons@mailandnews..com

specs

Compaq Computer Corp.
20555 SH 249
Houston, Texas 77070
Web: www.compaq.com
Phone: 281 370-0670

Specifications
Processor: 206 MHz Intel StrongARM 32-bit processor
Memory: 16MB RAM (H3135) or 32MB RAM (H3635)16MB ROM
Display:15 Grayscale STN (H3135) or 4086 Color TFT 240x320 resolution
Size: 5.11" x 3.28" x 0.62"
Weight: 6.3oz (H3135) or 6.0oz (H3635)
Power: Lithium Polymer Rechargeable battery or AC Adapter
Input: Touch-sensitive display, software keyboard, handwriting recognition, voice recorder, five application buttons, five-way joystick, on-off and backlight buttons and note taker
Ports: USB, CompactFlash expansion, infrared, microphone, speaker
Operating System: Windows CE (a version of Linux is available for the iPAQ – see www.pocketlinux.com)
Java Preinstalled: No. PersonalJava Runtimes are available from Sun, Insignia (Jeode), and others
Price: U.S.$399 (H3135) or U.S.$599 (H3635)
Notes: A 64MB RAM option is available; see the Compaq Web site for more details.

This is the first in a series of reviews of devices that are capable of running Java 2 Micro Edition – be it PersonalJava, MIDP, or any other new profile that comes along. In this and future issues of ***JDJ***, we'll try to provide a rundown of the various kinds of handheld, embedded, and mobile devices, and how well they run your favorite language. First up is Compaq's iPAQ.

The iPAQ, a pocket PC from Compaq, (see Figure 1), comes in two flavors. There's the gray-scale H3135, which is 5.1"x3.2"x0.6" and weighs in at 6.3 ounces, and the color H3635, which is the same size but 0.3 ounces lighter (according to the specs, at least). Both use the same processor, a 206MHz Intel StrongARM 32-bit chip;however, the H3635 has an extra 16Mb of memory.

The other important specs are a color LCD touch screen that supports 4,086 colors with a resolution of 240x320 pixels for the H3635; and a 15 gray-scale, semitransmissive STN LCD touch screen for the H3135. A CompactFlash Expansion slot is included in both, and extra RAM/modem options are available separately.

The iPAQ is one of the better looking PDAs. Considerable thought has gone into the design of the silvery case and it just looks good in the hand – and feels comfortable as well. A litmus test of its appearance is undoubtedly: Would you feel safe using the product in public? (That is, is someone likely to mug you and run off with it?) The iPAQ definitely rates as a device that little old ladies are likely to knock you senseless with their handbags just to get their hands on.

But enough about the boring hardware stuff. What we really want to know is the caffeine factor!

To use Java on any device when it hasn't been preinstalled will always be a hassle – usually it means a hefty download and then a problematic installation. Sun has a runtime environment for Java, available for the StrongARM processor (WinCE); however, it's currently in beta, so performance is less than stellar (based on my prior experience). Your average user probably isn't going to bother, which is disappointing, especially if you're developing applications and want to target the largest audience possible. It's more disappointing when you want to target the fastest processor currently available and aren't likely to get the performance your application requires.

However, Compaq has recently announced a $19.99 deal for users to purchase the JeodeRuntime for the iPAQ (see the Jeode review in this issue of ***JDJ***), so for the purposes of this review we'll be using the JeodeRuntime. And with ActiveSync, the installation just means downloading the runtime to the device.

When evaluating a Java-ready device, if it has a screen, only one thing really matters – graphics performance. Everything else is secondary. After all, you're not likely to be writing servlets for your handheld (although considering the perversity of some people in the development community, maybe you are, so scratch that as an argument). As the iPAQ doesn't come with a modem built in, network connectivity performance is of secondary importance. As the test model I was supplied with didn't come with the modem option, it's a nonstarter.

The tests we use are fairly simple; however, they do give us an idea of what we can expect from the iPAQ with more serious Java applications. In all cases, the tests are run (where graphical) using the full screen size available.

FIGURE 1 The iPAQ

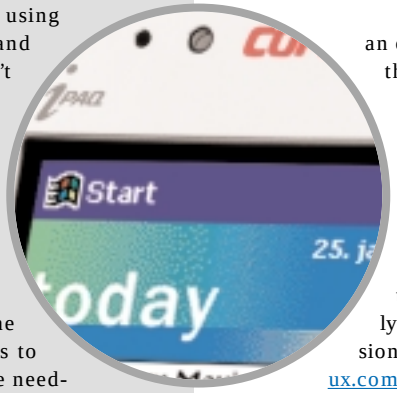
iPAQ TEST

The first test is an AWT application (or applet; it can run as either) with a few of the basic AWT components on a single screen (see Figure 2). What we want to know is how responsive are the components. When you click a button, is it slower to rebound than a snail on depressants? When you type (or write) a message into a text field, are you likely to sprout cobwebs under your arms before you see the text appear?

As expected, the news for the iPAQ is good – there's no noticeable sluggishness on any of the components with Jeode (the same is true for Sun's PersonalJava runtime). When entering text with the handwriting tool, it appears in the text boxes instantaneously; there are no perceptible problems with button, checkbox, or list drawing.

iPAQ TEST

The second test is basic graphics performance (see Figure 3). In this case, the test application uses one of the most fundamental graphics operations – drawLine() – to create random triangles on the screen. *Note:* The reason for using drawLine() to create the triangles, and not fillPolygon(), is that MIDP doesn't support the fillPolygon() method, and it's useful to use code in our PersonalJava test that's similar to what we might use in a MIDP test, for example. The application doesn't provide the frame rate, but rather the length of time to draw four random triangles on the screen.



In this case, the iPAQ (unlike the screenshot shown in Figure 3) starts to struggle a bit more: the average time needed to draw the four triangles is 450–800ms. Interestingly, in this test only, Sun's PersonalJava beats Jeode hands down, taking only a few milliseconds to complete the operation.

iPAQ TEST

The third test, if you're developing a game, is probably the most important: raw pixel throughput. In other words, if you provide an array of pixel colors, how fast can they be blitted to the screen as an image? This is useful if you want to write a game with more interesting particle effects, for example. This test application creates a random array of black and white colors, links the array to image creation, and paints the pattern to the screen as an image. Here we're interested in the maximum frame rate possible. Here the iPAQ produces an average frame rate of 10.6–10.8 frames per second – not quite as high as I hoped, but still an improvement on the Sun beta (around 1.3 fps).

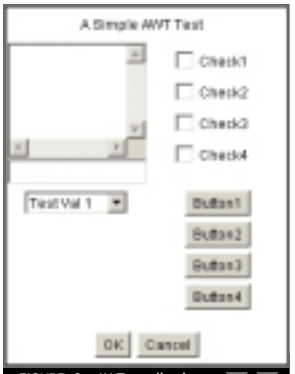


FIGURE 2 AWT application

FIGURE 3 drawLine()

Conclusion

Performance

With the JeodeRuntime, the iPAQ performs fairly well. It's good enough for minor animations, puzzle games, and the like, though perhaps not powerful enough for more advanced particle effects, 3D graphics, and more. Of course it would take more than a week's evaluation to really establish how the PDA fits in the gaming world, but certainly, for business applications, the power is more than adequate.

Secret Agent Factor

The iPAQ is a PDA, so secrecy isn't really an option while you're using it, and in terms of the competition, it comes in a little larger than a Palm Vx. However, it's an extremely sleek-looking piece of hardware – so a worthy addition to any budding spy's kit.

For the Gadget Geek

This device wins outright in the raw power stakes. But it's Windows CE, so it's got all those…Windows…bits and pieces. For really happy hacking, install an embedded version of Linux on it (for example, www.pocketlinux.com), and compile a version of Java for Linux. You'll probably destroy any chance you had of being truly productive with the device, but who cares! It's a PDA, it's Linux, it's Java… oh, just forget it!

Overall

From a Java developer's point of view, the iPAQ's processor is definitely a selling point. It looks good, has specifications that generally could put some laptops to shame, and coupled with Jeode, could be used to produce some fairly good multimedia applications (perhaps not cutting-edge, though). But bear in mind, if you're developing to take advantage of this extra power, your market may be fairly limited for a while; however, more and more devices are appearing with 200+ CPU speeds, so the market may not stay small for long.

References

1. *Compaq handhelds:* www.compaq.com/showroom/handhelds.html
2. *The Compaq/Jeode deal:* www.compaq.com/products/handhelds/java.html
3. *Jeode:* www.insignia.com/products/default.asp

The Missing Bits

A beginner's guide to writing applications for the MID profile

Part 2 of 3

WRITTEN BY
JASON BRIGGS

In Part 1 of this article, which can be found in JDJ (Vol. 6, issue 7), we covered the basics of creating a Mobile Information Device Profile application (also called a MIDlet). We covered some of the functionality available in the user interface packages and a slightly more advanced graphics example.

Missing from that discussion was one of the more unusual APIs included in MIDP – the Record Management System – which can be found in the package javax.microedition.rms. This is a persistent storage system based on a "simple record-oriented database." If you want to store data on a device (and not on the server, for example), you'll need to become very familiar with RMS.

The most fundamental object in RMS is the javax.microedition.rms.RecordStore, which is owned by a particular suite of applications currently installed on the device. What this means is that all the applications in the suite can share the same data, if necessary – games, for example, can store their saved state in one RecordStore. A phone book application might share its data with a simple scheduling application – as long as it's in the same suite, of course.

To open (or create) a new RecordStore, use a static method found in the RecordStore class. Assuming we wanted to write a Phone Book MIDlet, we would probably call our store "PhoneBook".

```
RecordStore store =
RecordStore.openRecordStore("PhoneB
ook",true);
```

The Boolean value "true" is used to specify that the store will be created, if necessary.

A RecordStore has a number of methods that can be applied to it. The most useful of these are:

- ***addRecord(…):*** Add a new record to the store
- ***deleteRecord(…):*** Delete a record from the store
- ***getNumRecords(…):*** Get the number of records in the store
- ***getRecord(…):*** Get a copy of data in a record
- ***getSize():*** Get the size of the record store (in bytes)
- ***getSizeAvailable():*** Get the available space for the store to grow (also in bytes)
- ***setRecord(…):*** Set the data in a specified record

Enumeration Is Not the Same as Remuneration

Use a RecordEnumeration to move back and forth through the store and view the data:

```
RecordEnumeration recordEnum =
store.enumerateRecords(null,
pbcomp, false);
```

Where the first argument (null) is the filter that specifies the subset of records that will be enumerated, the second argument (pbcomp) is a RecordComparator (more on that soon) that specifies the order, and the final argument (false) specifies that the enumeration will, in this case, not be kept up-to-date with the contents of the store (in other words, the enumeration is a copy of the store at that point).

A RecordComparator, as mentioned before, is used to order an enumeration. Create a comparator by implementing the RecordComparator interface. The interface describes a single method (compare) that returns an integer value depending upon whether the contents of one byte array FOLLOWS, PRECEDES, or is EQUIVALENT to a second byte array (see Listing 1). Listings 1–7 can be found on the *JDJ* Web site, www.javadevelopersjournal.com.

Now that we have an enumeration of records, what can we do with it? The important methods in the RecordEnumeration are:

- ***hasNextElement():*** Does the enumeration have more elements?

FIGURE 1 Main menu

FIGURE 2 Screen

FIGURE 3 Entry Screen

FIGURE 4 List of tests

FIGURE 5 Output from tests

- ***hasPreviousElements():*** Are there elements prior to the current position?
- ***nextRecord():*** Get a copy of the data in the next record
- ***nextRecordId():*** Get the recordId of the next record
- ***numRecords():*** Get the number of records in the enumeration
- ***PreviousRecord():*** Get a copy of the data in the previous record
- ***PreviousRecordId():*** Get the recordId of the previous record

In the case of our aforementioned PhoneBook application, we might display the contents of the phone book on a canvas and use the up and down buttons to scroll back and forth through the enumeration dataset. On the canvas we could trap the key press and then pass the game action (game actions are special events, such as pressing an arrow button, which are mapped to particular keys) back to the parent application (containing the data store) to handle:

```
public void keyPressed(int k) {

parent.processAction(getGameAction
(k));
}
```

In the parent class (in this case the MIDlet), the processAction() method will move forward and backward in the enumeration depending upon whether the up or down arrow has been pressed. For example:

```
if (action == Canvas.UP &&
recordEnum.hasPreviousElement()) {
  currentRecordID =
recordEnum.previousRecordId();

set(store.getRecord(currentRecordI
D));
}
else if (action == Canvas.DOWN &&
recordEnum.hasNextElement()) {
  currentRecordID =
recordEnum.nextRecordId();

set(store.getRecord(currentRecordI
D));
}
```

In this case, currentRecordID is an int that holds the recordId the enumeration is currently pointing to, and set(…) is a user-defined method that calls the canvas and changes the display accordingly.

Listing 2 provides the full method source; Listing 3 provides the source of the canvas class.

Large Mouthfuls, or Multiple Bytes

The records in a store are byte arrays, meaning you store and retrieve your records as simple byte arrays. So you can either work with the raw bytes (as I've done in my PhoneBook application, for simplicity's sake), or use a combination of ByteArray and data streams. Obviously, unless you have a requirement for fixed data sizes (which is inefficient, but again that's what I'm using in the PhoneBook), the stream combination will probably be the more efficient way to go, in terms of the most effective use of storage space.

It has the added advantage that using the various read methods in a DataInputStream (for example, readInt, readLond, readShort, readUTF), keep you one step away from the messy details of how that data is actually stored in the byte array.

To use the stream combination, you'll probably do something like the following:

```
ByteArrayOutputStream baos = new
ByteArrayOutputStream();
DataOutputStream dos = new
DataOutputStream (baos);
dos.writeUTF("Hello");
dos.writeUTF("Test");
dos.writeInt(100);

byte rec[] = baos.toByteArray();
```

In comparison, the save() method in PhoneBook.java uses a fixed length name and variable length phone number, creating byte-arrays of the data using a specially defined function called *rpad* (which adds spaces on the end of the name if it's not long enough) (see Listing 4).

I leave it up to the individual to decide which way works better for them. One of the joys of Java is that while it provides functionality to insulate developers from any "nastiness," it also allows you to get in there and "do things your own way."

Figures 1–3 show the main screens in the PhoneBook MIDlet.

Networking: It's More than Making Friends and Influencing People

MIDP has to include networking support. A large number of Java developers in lynch-mode, with nooses made from coaxial cable and baseball bats molded out of melted-down network cards, would turn up on Sun's

doorstep in Palo Alto baying for blood if a Java product was released without adequate network support, and Sun knows that. Accordingly, you can find the various network IO classes in the javax.microedition.io package. However, network connectivity is handled differently from what you might be used to in the Standard Edition of Java (in java.net.*). One major difference is that instead of creating/opening a socket or a URLConnection and then getting an input stream from that object, you use the Connector class to create and open a Connection, an Input/OutputStream, or a DataInput/DataOutputStream. For example:

```
HttpConnection c =
(HttpConnection)Connector.open("ht
tp://.....");
```

There are a number of Connection interfaces defined in javax.microedition.io: Connection, ContentConnection, DatagramConnection, HttpConnection, InputConnection, OutputConnection, StreamConnection and StreamConnectionNotifier.

In Listing 5 we'll look at the HttpConnection interface using a MIDlet called HttpTest.java. In this application we'll define a test method that takes a URL string as a parameter and then opens an HTTP POST connection to that URL (for more information on HTTP see RFC 2616 – www.ietf.org/rfc/rfc2616.txt?number=2616).

Looking at Listing 5 line by line: Lines 1 and 2 declare the HttpConnection object and the InputStream. Line 5 opens an HttpConnection to the supplied URL. Line 6 sets the request method on the connection. In this case we want to perform HTTP POST operations. Line 8 opens an input stream to the connection. Line 10 tries to get the content length. If a content length is provided, lines 12 and 13 create a byte array and then read from the stream into the array. If no length was provided, lines 18–22 read from the stream, character by character, until a terminator is reached. Lines 15 and 23 return the result from either set of operations. Finally, lines 27–30 and 32–35 close the InputStream and Connection.

I think you'll agree that this is fairly painless networking (as with most communications in Java).

What else is included in this MIDlet? HttpTest creates a simple list of five tests that's used as the first screen the user sees. When the user selects a test, and then selects the OK command, the code in Listing 6 is executed.

In Listing 6 lines 2–5 destroy the application if the Exit command was selected. Line 7 checks the current state, which is used to switch between the two screens, and line 8 changes the display to a TextBox (tb), which will be used to show the output. Line 11 gets the currently selected index in a list of tests that can be executed. Line 13 checks that the test selected is valid. Line 14 calls the test method with the URL of a TestServlet and passes a parameter action, which will be in the form, http://localhost:8080/servlet/TestServlet?action=n (*n* being the numbers 1-5, depending upon the user's selection). Lines 16–18 check that the test method returned a valid result, and assigns that result to the output TextBox, using the setString(...) method. Finally, lines 26–28 change the display back to the list of available tests.

It Takes Two

One other thing is missing from this network conversation – TestServlet.java, shown in Listing 7. This is not the right place for a deep and meaningful discussion on servlet development, but to summarize what the servlet is doing:

1. A doPost event arrives to be handled by the servlet.
2. The servlet gets the writer for the ServletResponse object.
3. The content type of the response is set.
4. The parameter "action" value is retrieved from the ServletRequest.
5. The parameter is converted to an int, and the word "test" is appended to a string a number of times based upon that number.
6. Finally, the number is concatenated on the end of the string, and the result is written out using the writer.

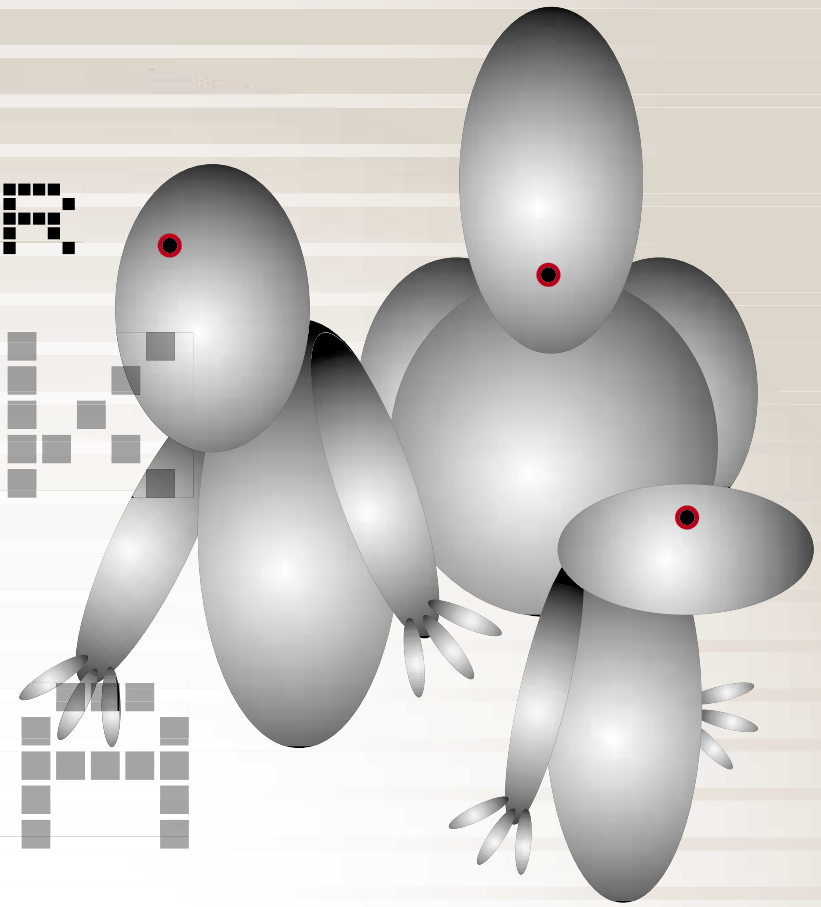
Figures 4 and 5 show the MIDlet (and servlet, behind the scenes) in action:

Summary

We've now covered the major parts of creating MIDP applications. In Part 3, we'll put all the pieces together extending the PhoneBook application already created. We'll retrieve data using an EJB, and funnel it through a servlet and into our J2ME front end.

jasonbriggs@sys-con.com

A DEEPER LOOK AT JAVA



Anatomy of a Java program

Written by Jacquie Barker

This excerpt discusses the specifics of coding the Student Registration System (SRS). Java is an extremely rich language, and our goal is not to duplicate the hard work that has gone into existing Java language books, but rather to complement them by showing you how to bridge the gap between producing an object model and turning it into live Java code.

Setting Up a Java Programming Environment

Everything you'll need to get started programming in Java on various platforms – Solaris, Windows, Linux – is available for free with Sun Microsystems' Java 2 Software Developer's Kit (SDK), which can be downloaded from Sun's Web site, http://java.sun.com. We strongly advise that you take the time now to establish your Java development environment, so you'll be prepared to experiment with the language as you learn.

Note that the Java 2 SDK is a command line–driven toolkit, which means that on a Windows platform you'll be opening up an MS-DOS Prompt window, from which you'll be doing all of your work (UNIX and Linux are naturally command-line oriented). Of course, there are also numerous Java Integrated Development Environments (IDEs) to choose from, some of which are available on a free trial basis as Web downloads. However, my personal bias is that if you first learn Java from the ground up, writing all your code from scratch using only Sun's SDK and your favorite text editor, you'll gain a much better understanding of Java language fundamentals than if you rely too heavily on an IDE, particularly those that provide drag-and-drop, GUI-building capabilities and automated code generation. You can always consider graduating to an IDE after you've mastered the language to take advantage of their debugging and code/project management features, among others.

Anatomy of a Java Program, Revisited

The anatomy of a trivially simple Java program, which consisted of a main() method, contains the logic of our program inside of a class "wrapper" (see Figure 1).

Such a class would reside in a file by the name of classname.java; Simple.java, in this example.

Note: The external name of the file containing the source code of a Java class must match the name given to the class inside the file, including the same use of upper/lowercase, with .java tacked on at the end. Java is a case-sensitive language, even on operating systems like DOS that are traditionally case insensitive in most respects.

A common error for beginners is to assume that case doesn't matter, and to name the file for our example program simple.java, SIMPLE.java, or some other variation. This leads to compilation problems, as we'll see in a moment.

A nontrivial Java application consists of many such .java files, because the source code for each class comprising your application typically (but not always) resides in its own *.java file.

You'll have one .java file for each of the domain classes that you defined in your object model. For the SRS application, for example, we'll have eight. Course.java, Person.java, Professor.java, ScheduleOfClasses.java, Section.java, Student.java, Transcript.java, and TranscriptEntry.java.

You'll also typically have a separate .java file for each of the primary "windows" comprising the graphical user interface of your application, if any. For the SRS application, we'll have

This is an excerpt from Chapter 13 of *Beginning Java Objects* written by Jacquie Barker, published by Wrox Press.

two: MainFrame.java and PasswordPopup.java.

Typically you'll have a separate .java file that contains the "official" main() method that serves as the application driver. One of the primary responsibilities of this driver class's main() method is to instantiate the core objects needed to fulfill a system's mission; of course, actions taken by the objects as well as by the user will cause additional objects to come to life over time, as the application executes. The main() method is also responsible for displaying the start-up window of the graphical user interface of an application, if any (see Figure 2).

We'll name the driver class for our Student Registration System application SRS, so of course it will need to be stored in a file named SRS.java.

Finally, you'll quite possibly have other "helper" classes necessary for behind-the-scenes application support; with the SRS, we'll have a need for three such classes: CollectionWrapper.java, CourseCatalog.java, and Faculty.java.

Assuming that you've properly installed Sun's Java 2 SDK, a Java source code file (.java file) can be compiled at the command line via the command:

```
javac classname.java
```

for example:

```
javac Simple.java
```

Again, pay close attention to match upper/lowercase usage; if you were to name your class Simple (uppercase "S"), but store it in a file named simple.java (lower case "s"), the Java compiler would generate the following compilation error:

```
Public class Simple must be defined in a file called
'Simple.java'.
```

Note that you can compile multiple files in a single step:

```
javac file1.java file2.java ... filen.java
```

or

```
javac *.java
```

Assuming that no compilation errors occur, compilation produces at least one classname.class file for every .java file; for example, Simple.class for our sample program (see Figure 3). (We'll see later in this article why more than one .class file might be produced from a single .java file.) The *.class files contain platform-independent Java byte code that can be run on any platform for which there is a Java Virtual Machine available.

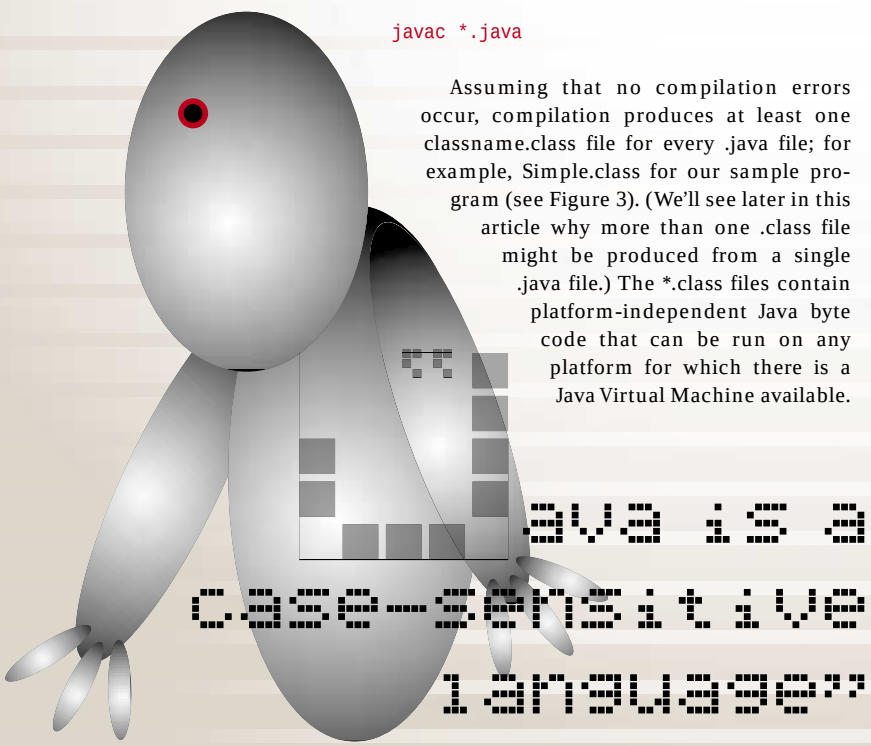


FIGURE 1 A simple Java program

To run a Java program from the command line, type the following to invoke the JVM:

```
java MainClassName
```

for example:

```
java Simple
```

or

```
java SRS
```

where MainClassName is the name of the class file (minus the .class suffix) containing the compiled byte code version of the "official" main() driver method for your application.

The JVM loads the byte code for whatever class you've named, and if it discovers a main() method within that byte code with the proper signature (recall that the name of the argument being passed into the main() method – args, in this case – is the only thing that's flexible in the signature):

```
public static void main(String[] args)
```

then the JVM executes that main() method to jump-start your application. From that point on the JVM will load additional classes – either classes that you've written and compiled or classes built into the Java language – as needed, when referenced by your application. That is, the first time the SRS application has occasion to refer to the Person class, the byte code for the Person class will be loaded into the JVM, and so forth.

It's important that you don't type the .class suffix when attempting to run a program, as you'll get an error:

```
java Simple.class

Exception in thread "main"
java.lang.NoClassDefFoundError: Simple/class
```

which is, to say the least, not very intuitive! This particular error message arises because the Java compiler interprets the name Simple.class as being the name of a class called "class", to be found within a package called "Simple"; we'll be talking about packages shortly.

Why must the main() method of an application be declared static? At the moment that the JVM first loads whatever class you've told it to load to jump-start your application, no objects exist yet, because the main() method hasn't yet executed; and it's the main() method that will start the process of instantiating your application's objects. So, at the moment of application start-up, all the JVM has at its disposal is a class; and a static method is a type of method that can be invoked on a class as a whole, even if we don't have an instance of that class handy.

One final note about program structure: we said that the source code for each class comprising your application typically resides in its own .java file. It's actually permissible to place the source code for two or more Java classes back to back in the same physical .java file. We don't generally do so, however, as it's much easier to manage Java source code when there is a one-to-one correspondence between the external file name and the internal Java class name. If we were to combine multiple class definitions back-to-back in a single .java file, however, they would each produce their own .class file when compiled.

Importing Packages

To appreciate import statements (see Figure 4), we first must understand the notion of Java packages.

Because the Java language is so extensive, its various built-in classes are organized into logical groupings called *packages*. For example, we have:

- ***java.sql:*** Contains classes related to communicating with Object Database Connectivity–compliant relational databases
- ***java.io:*** Contains classes related to file input/output
- ***java.util:*** Contains a number of utility classes, such as the Java collection classes that we'll be learning about
- ***java.awt:*** Contains classes related to GUI development

Most built-in Java package names start with java, but there are some that start with other prefixes, such as javax. If we acquire Java classes from a third party, they typically come in a package that starts with com.companyname, for example, com.xyzcorp.stuff.

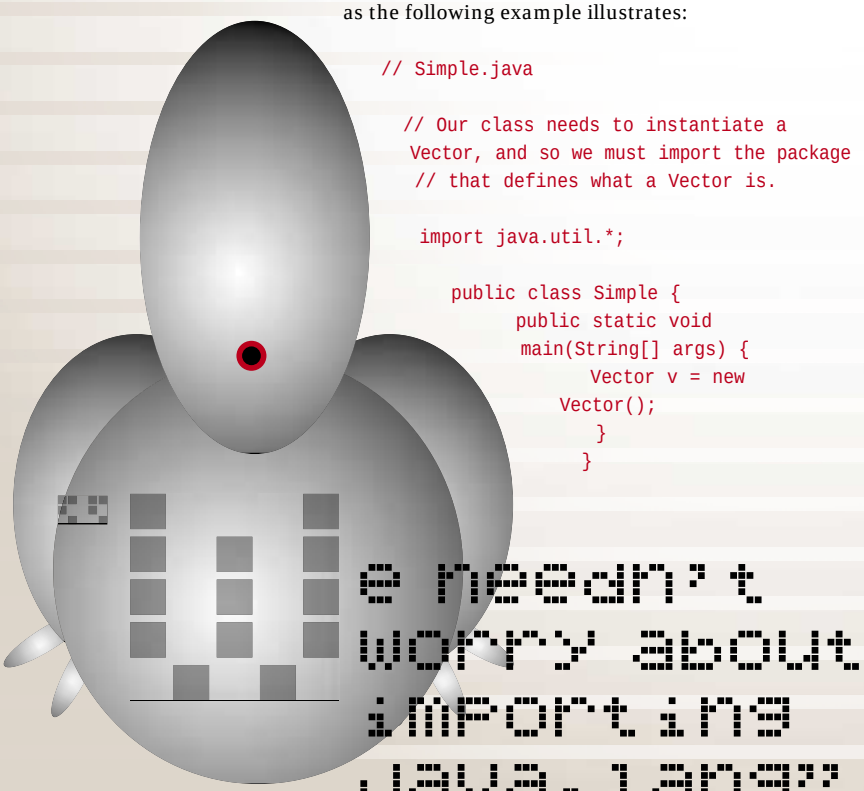
The package named java.lang contains the absolute core of the Java language, and the classes contained within that package are always available to us whenever we write Java programs, so we needn't worry about importing java.lang. However, if we wish to instantiate a Vector (one of Java's built-in collection classes) as an attribute inside one of our classes, for example, then we must import the java.util package, as the following example illustrates:

```
// Simple.java

  // Our class needs to instantiate a
   Vector, and so we must import the package
   // that defines what a Vector is.

    import java.util.*;

       public class Simple {
             public static void
              main(String[] args) {
                    Vector v = new
                Vector();
                   }
                 }
```



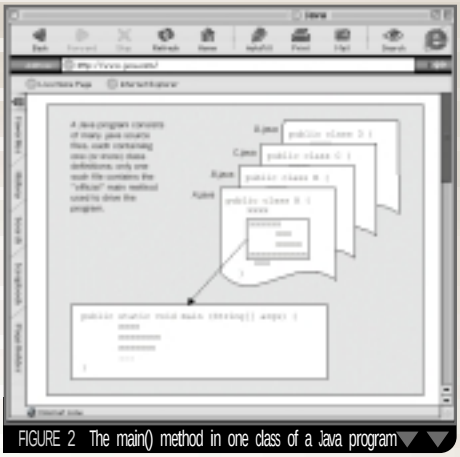


FIGURE 2 The main() method in one class of a Java program

FIGURE 3 Creation of multiple class files from multiple Java files

The asterisk (*) at the end of the import statement above is a wild card character; it informs the Java compiler that we wish to import all of the classes contained within the java.util package. As an alternative, we can import individual classes from a package:

```
// ImportExample.java

// We can import individual classes, to better docu-
ment where each class
// that we are using originates.

import java.util.Enumeration;
import java.util.Vector;
import java.util.Date;
import java.io.PrintWriter;

// etc.
```

Of course this requires more typing, but it serves as better documentation of where each class that we're using in our program originates.

If we were to attempt to reference the Vector class in one of our classes without this import statement, we'd get the following compilation error when compiling that particular class:

```
Class Vector not found.
```

This is because Vector is not in the name space of our class; that is, it's not one of the names the Java compiler recognizes in the context of that class. Generally speaking, the name space for a given class contains the following categories of names, among others:

- The class
- All the attributes of the class

Essential Technologies for Today's B2B Integration Challenges

Web Services EDGE

PRE-REGISTER ONLINE TODAY
SAVE $300

SEPTEMBER

New York, NY

web services conference&expo EDGE EAST COAST — 24-25

JDJ EDGE conference&expo — 23-26

OCTOBER

Santa Clara, CA

web services conference&expo EDGE WEST COAST — 24-25

XML EDGE conference&expo — 22-25

Each venue features:

- Over 100 Information-Packed Sessions
- Two Distinct Conferences for Just One Registration Fee
- Pioneers and Visionaries
- Leading Vendors Presenting New and Problem-Solving Products

ColdFusion fasttrack
SEPT. 24-25, 2001
TWO
CONCENTRATED DAYS OF LEARNING TO JUMPSTART YOUR PROFESSIONAL CREDENTIALS!

Participating Companies, Sponsors and Exhibitors as of June 1, 2001

TogetherSoft, SilverStream, sitraka software, sonic software, aligo, KENAMEA, shinka technologies, SAVAJE, INFERDATA, bea, InCert, SEAGULL, Data Quad, idoox, CAPE CLEAR, ParaSoft, MERANT, PointBase, hp invent, pingtel, TopCoder, NOMA LABS, Bestning, Rational, ROGUE WAVE software, OASIS, ORACLE, DataMirror, CodeHound.com, DATA JUNCTION, ENTIRETY Solutions, TOWER, xml spy, TALKING BLOCKS, WHITEHILL, AdvanTel, XML.ORG, XMLCities, XML Journal, Programmers Paradise, ColdFusion Developer's Journal, WebSphere, WebServices, WebLogic, webtechniques, JAVA Developer's Journal, SD Times, wireless

Secure Your Seat Now!

Two Easy Ways to Register

1 On the Web:
www.SYS-CON.com

2 By Telephone:
201 802-3069

Be sure to visit www.SYS-CON.com for travel information.
Reserve Your Hotel Room Early!

- All the methods of the class
- Any local variables declared within a method of a class
- All classes belonging to the package that the class in question belongs to
- All public classes in any other package that have been imported
- All public features (attributes, methods) of any of the classes whose names are in the name space
- All public classes in java.lang

We could work around the failure to import a package by fully qualifying the names of any classes, methods, or other, that we use from such a package; that is, we can prefix the name of the class, method, or other, with the name of the package from which it originates, as shown in the next example:

```
// Simple2.java

// no import statement

public class Simple {
  public static void main(String[] args) {
    java.util.Vector v = new java.util.Vector();
  }
}
```

This, of course, requires a lot more typing, and impairs the readability of the code.

Although most built-in Java packages have names that consist of two terms separated by periods (for example, java.awt) some built-in Java packages have three, for example, java.awt.event. As far as packages developed by third-party organizations are concerned, there's really no limit to the number of terms that can be concatenated to form a package name. The important point to note about all of this is that the statement:

```
import nameA.nameB.*;
```

will only import classes in the nameA.nameB package; it won't import classes in the nameA.nameB.someothername package. That is, the wild card pertains to class names only.

It's also important to note that importing a package is only effective for the particular .java file in which the import statement resides. If you have three different classes of your own that all need to manipulate Vectors, then all three of their .java files must include an import statement for java.util.

Java also provides programmers with the ability to logically group their own classes into packages. If we wanted to, we could invent a package, such as com.objectstart.srs to house our SRS application. Then, anyone else wishing to incorporate our SRS classes within an application that they were going to write could include the statement:

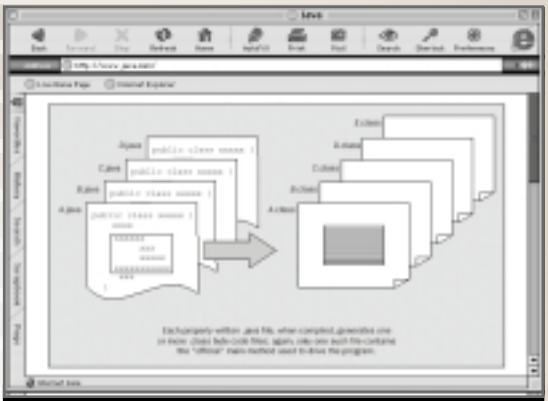


FIGURE 4 Modified Java file

```
import com.objectstart.srs;
```

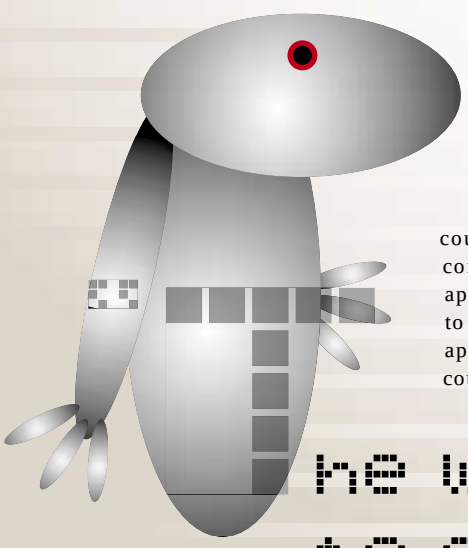
in their code, and even though our compiled class files are kept physically separated from their application's compiled class files, our classes would become logically combined with theirs, assuming a few other environmental details had been taken care of.

Going into a detailed discussion of how to create our own packages is beyond the scope of this article. But, as it turns out, if you do nothing in particular to take advantage of programmer-defined packages, then as long as all of the compiled .class files for your code reside in the same directory on your computer system, they're automatically considered to be in the same package, known as the default package. All the code that we write for the SRS application will be housed in the same directory, and hence will fall within the same default package. This is what enables us to write code such as:

```
public class SRS {
  public static void main(String[] args) {
    Student s = new Student();
    Professor p = new Professor();
    // etc.
  }
}
```

without using import statements, because Student, and Professor, and SRS, and all of the other classes comprising our SRS applications are within the same default package.

The bottom line is that import statements as a building block of a .java source code file are optional; they're needed only if we're using classes that are neither found in package java.lang nor in our own (default) package.



"The wild card pertains to class names only"

ABOUT THE AUTHOR

Jacquie Barker is a professional software engineer and adjunct faculty member at George Washington University and Johns Hopkins University and a principal member of the technical staff at SRA International, Inc. in Fairfax, Virginia. She holds a BS in computer engineering from Case Western Reserve University and an MS in computer science, from UCLA.

jjbarker@objectstart.com

What's Online...

August **2001**

JDJ Online

Check in every day for up-to-the-minute news, events, and developments in the Java industry. Can't get to the newsstand on time? Just visit www.javadevelopersjournal.com and be the first to know what's happening in the industry. Check out what you've been missing.

Subscribe to Our FREE Weekly Newsletters

Now you can have the latest industry news delivered to you every week. **SYS-CON** newsletters are the easiest way to keep ahead of the pack. Register for your FREE newsletter today! There's one for Java, XML, Web Services, Wireless, and ColdFusion. Choose one or choose them all!

JavaOne 2001 Radio Interviews

Tune in to **SYS-CON Radio** (via the Shoutcast Network) and hear the industry's top movers and shakers, interviewed by ***JDJ*** editors. Listen to James Gosling, the father of Java, ***JDJ*** Advisory Panel members, CEOs, product managers, and other top executives as they share their insight and opinions on today's hottest issues.

JDJEdge Conference Program Now Available; Opening Day Keynotes to be Delivered by James Gosling and Alan Baratz

Join us at the Hilton New York, September 23 – 26, in New York City for the largest Java and Web Services event on the East Coast.

James Gosling and Alan Baratz will deliver the opening day keynotes; top executives from Sun, IBM, BEA, Macromedia, PointBase, Sitraka, and Zaplet are also scheduled to speak.

The conference offers cutting-edge tracks, night school, and an accelerated weekend program. More information about the conference program and registration is available at www.sys-con.com/javaedge/.

Product Review

Considering a product upgrade? Want to know the ins and outs of a new product before you purchase it? Then make sure you read our in-depth product reviews.

Our writers test and evaluate a host of Java products to aid your decision-making process. We get behind the hype and give you the facts. All reviews are written by experts and leaders in the information technology industry.

JavaDevelopersJournal.com Developer Forums

Join our new Java mailing list community. You and other IT professionals, industry gurus, and ***Java Developer's Journal*** writers can engage in Java discussions, ask technical questions, talk to vendors, find Java jobs, and more. Voice your opinions and assessments on topical issues – or hear what others have to say. Monitor the pulse of the Java industry!

RadView Launches WebLOAD 5.0

(*Burlington, MA*) – RadView Software Ltd. recently announced WebLOAD 5.0. The latest version offers up to a 60% performance enhancement over previous versions and enables software developers and QA professionals to more easily and efficiently verify the scalability of e-business apps throughout the development life cycle. www.radview.com

realMethods Releases Framework 1.0

(Bridgewater, MA) - realMethods, Inc. has announced the release of the Framework 1.0, a robust system designed for speeding development of the most difficult J2EE apps. The realMethods Framework is based on the most widely recognized J2EE Design Patterns and Sun Microsystems' J2EE Blueprints. A downloadable version is available at www.realmethods.com.

Performance Benchmarks Demonstrate New Release of FioranoMQ

(*Los Gatos, CA*) – Fiorano Software, Inc. announced benchmark results that demonstrate that the latest version of FioranoMQ is more than 100% faster than the previous version released in February 2001.

The performance benchmarks consist of 72 individual tests in varying real-world scenarios. A significant optimization of the message headers as well as improvements to the unique file-based data store were the main reasons behind the performance improvement.

The performance tests used for the benchmarks are publicly available from the Fiorano Web site at www.fiorano.com.

Devicetop.com Announces Developer Contest

(*Ontario, Canada*) – Devicetop.com has recently announced its third developer contest for smart device applications. In phase 1, contestants submit proposals for applications for interactive digital TVs, Internet appliances, and automotive infotainment systems. Six winners of the first phase will receive $2500. These six ideas will be used for phase 2 of the competition, which will be to build one of the applications. For more information on the Devicetop competition, see www.devicetop.com.

Applied Reasoning Introduces Mobile Classic Blend

(*Overland Park, KS*) – Applied Reasoning has announced Mobile Classic Blend, a Java middleware product that delivers fast Internet wireless applications on Java-enabled mobile devices. The current release supports the Palm OS platform on the Kyocera Smartphone and uses IBM's J9 virtual machine on the Smartphone. www.appliedreasoning.com

Eoscene Announces World's First J2EE Based OLAP App

(*Seattle, WA*) – Eoscene Corporation has announced the rollout of the world's first J2EE compliant On-Line Analytical Processing (OLAP) application, as part of its Intelligence Portal product.

Eoscene's new technology gathers and interprets data from any structured data source based on its Meta models, combines several data sources, and saves the data in its OLAP store that runs on most RDBMSs including Oracle, Informix, SQL Server, DB2, Sybase, and PostgreSQL. www.eoscene.com

FastObjects Partners With Wind River

(*San Mateo, CA*) – FastObjects by Poet has joined the WindLink Partner Program of Wind River Systems, Inc. Through the WindLink Partner Program, FastObjects is making its embedded object database technology available to customers building systems based on Wind River's VxWorks real-time operating system (RTOS). www.windriver.com www.fastobjects.com

CA Delivers Unicenter 3.0

(*Islandia, N.Y.*) – Computer Associates International, Inc. has announced the general availability of Unicenter 3.0 Network and Systems Management (NSM), the cornerstone of its new Unicenter family of solutions. New features include enhanced dynamic 2D and 3D visualization, personalization and root-cause analysis: technologies. http://ca.com

Aligo Offers Free Web Seminars

(*San Francisco, CA*) - Aligo, a leading wireless software infrastructure company is offering live Web seminars on the business benefits of mobile apps and Java. Classes are under one hour. www.aligo.com

SYS-CON Media Named Winner of the 2001 New Jersey Technology Fast 50 Awards

(*Montvale, NJ*) – **SYS-CON Media** (www.sys-con.com), the world's leading *i*-technology publisher and the producer of *i*-technology developer conferences, headquartered in Montvale, NJ, has been named by Deloitte & Touche to its annual list of the 50 fastest-growing technology companies in New Jersey. As a winner of the Technology Fast 50 Award, **SYS-CON** was also nominated for the national competition for the 500 fastest-growing technology companies in the United States, the Technology Fast 500 Awards, with results to be announced later this year.

In 1999, **SYS-CON Media** was ranked 194 by *Inc.* 500 on the list of America's fastest-growing privately owned companies. **SYS-CON** was nominated again for *Inc.* 500 in 2001 and is awaiting the results. The 2001 ranking of Technology Fast 50 is based on percentage of corporate revenue growth over the five-year period between 1996 and 2000. Winners include both public and private corporations.

"We are delighted to be a 2001 New Jersey Technology Fast 50 winner," said Fuat Kircaali, founder and CEO of **SYS-CON Media**. "Although we have experienced consistent growth since 1994, the year the company was founded, only after **SYS-CON** turned five years old were we eligible for these nominations. This year, based on our 2000 operating results, we are confident that our company will be named as one of America's fastest-growing companies once again by *Inc.* 500, as well as by Deloitte & Touche Technology Fast 500." In 2000, Fuat Kircaali was nominated by Ernst & Young for their Entrepreneur of the Year award.

Technology Fast 50 winners will be honored at an annual awards breakfast on August 16, 2001. Winners of last year's New Jersey Technology Fast 50 included IDT Corp., Programmer's Paradise, Novasoft, Neil Laboratories, Computer Horizons, and Hexaware Technologies.

SYS-CON received the Technology Fast 50 award based on a 1,075% five-year revenue growth between 1996 and 2000. **SYS-CON**'s prior national recognition, the 1999 *Inc.* 500 award was given based on a 1,307% five-year revenue growth between 1994 and 1998.

The company's first national recognition, published in *Inc.* magazine's October 1999 special *Inc.* 500 issue and reaching over 2.5 million readers, was an exclusive report on the companies and CEOs who are changing the face of American business. Noteworthy alumni include household corporate names such as: Microsoft, Oracle, WordPerfect, CompUSA, Gateway 2000, Intuit, e*Trade, Timberland, Jenny Craig, and Domino's Pizza.

Who Needs an Agency?

Do they provide valuable services?

WRITTEN BY
BILL BALOGLU & BILLY PALMIERI

You're a skilled technical professional with experience that's in demand. Your résumé reads like a Who's Who of top companies and a What's What of top skills. So how come you're the one working 40 hours a week, but a chunk of the money the client is paying for your services is going or has gone to the agency that placed you?

If you've been placed in a permanent position, why should the client pay your agency a percentage of your annual salary for just sending out your résumé and setting up an interview? Or if you're a contractor, sure, the agency sends you a timesheet and mails you your paycheck, but how much does that cost? And how much is the client company you're working for paying them?

At some point these questions are bound to run through the mind of any intelligent engineer. No one wants to feel like they're being taken advantage of, or share their income with a company that's a necessary evil.

There are good agencies out there. They provide valuable services to engineers, not the least of which is finding you that opportunity in the first place.

But with the past few years' boom in demand for technologists, innumerable agencies have sprung up, eager to cash in on the high-tech bandwagon without much regard for little details, such as industry experience, technical expertise, or solid business ethics.

What should you look for in an agency? What should you expect from them? And what should they expect from you? Our combined experience on both sides of the technologist/agency relationship has given us more than a few insights that we'd like to share.

The Technologist/Agency Relationship

"Traditionally speaking, many agencies represent a broad spectrum of technical professionals. The trend is changing, however," says Kiran Khanna, a senior recruiter at ObjectFocus. "There is a strong demand for the services of specialized, niche agencies that can provide the needed talent in specific technology segments quickly and efficiently."

Many potential candidates balk at providing references to an agency, claiming, "I work with lots of agencies, I can't have all of them bothering my references." This leads us to a very important distinction between reputable and "fly-by-night" agencies.

When you get a phone call from a fast-talking recruiter who can't wait to zap your résumé off to a client for "a great opportunity," how carefully does that recruiter explain the position? Unfortunately, the majority of agency recruiters know little, if anything, about the kind of work you do.

Their primary goal is to spam out as many résumés as possible (often unsolicited) to clients who may be looking for someone with skills that are totally different from your own.

And your résumé arrives on the desk of an annoyed, frustrated hiring manager with that agency's letterhead emblazoned across the top. What many engineers don't realize is the following:

- ***You're guilty by assocation.***

These agencies that are mostly interested in closing a quick deal won't bother to fully qualify you before submitting you for a position. You can bet that they're also submitting totally unqualified candidates to their clients.

The agency has no credibility with the client and therefore you have no credibility with the client. You have only partnered with that quick-sale agency to waste the hiring manager's valuable time. And burn a potential bridge.

- ***A good agency has strong working relationships with its clients.***

When approached by an agency, ask the recruiter about the relationship with the client. How long have they been providing candidates to that client? How many of their consultants are currently working with that client? What is the hiring manager like? What is the style and culture of the company?

If the recruiter can't answer at least some of these questions, there's a good chance they have no relationship with that client at all. Because they lack credibility, many quick-sale agencies are merely trying to fill positions they pulled off a company's Web site.

- ***Show me the money!***

So you've found yourself a good, solid agency to work with that cares about building a long-term relationship with you. You've found your Jerry Maguire. What does that agency do to earn its percentage?

Building and maintaining strong relationships with new and existing clients requires constant effort, from cold-calling to seven-day-a-week networking. The end result is the job you're working on today and in the future.

Each client has its own contract to be negotiated, which means detailed, time-consuming legal work before you begin your new position. If you're on a contract, there's constant monitoring of your project, progress, and on-site relationships enabling the agency to solve problems before they occur.

By constantly searching for and qualifying new engineers, agencies burn up the phone lines and ever-changing Internet resources.

A good agency should provide full disclosure of their fees and margins, regardless of whether it's a contract or permanent placement. Not all agencies do this, but at the very least, they should be willing to tell you the placement fees or the percentage markup they're charging.

As in any business, there's no greater source of credibility than a referral. Once an agency has said they'd like to work with you, ask to speak to one of the engineers who currently works with them. You're likely to get valuable insight.

AUTHOR BIOS

Bill Baloglu is a principal at ObjectFocus (www.ObjectFocus.com), a Java staffing firm in Silicon Valley. He was a software engineer for 16 years prior to his position at ObjectFocus. Bill has extensive OO experience, and has held software development and senior technical management positions at several Silicon Valley firms.

Billy Palmieri is a seasoned staffing industry executive and a principal of ObjectFocus. Before that he was at Renaissance Worldwide, a multimillion-dollar global IT consulting firm where he held several senior management positions in the firm's Silicon Valley operations.

billb@objectfocus.com

billp@objectfocus.com

Radically Iconoclastic

WRITTEN BY
BLAIR WYMAN

The powers that be let me out of my cube long enough to attend the JavaOne Conference a couple of months ago in San Francisco. While I was there, I got the chance to meet some of the movers and shakers behind *Java Developer's Journal*. Of course, I had already met a couple of the "big cheeses," but by no means all of them.

At one point I was introduced as ***JDJ***'s "resident iconoclast." Naturally, I smiled a big smile, puffed out my chest, and tried my best to look genuinely proud of myself. Then, at the first opportunity, I made some excuse to steal away and look up the word "iconoclast."

Well, as it turns out, my feigned pride was a good guess. It seems that the term *iconoclast* historically originated with wild-eyed 8th-century Byzantine Christians destroying religious images and idols – and later came to mean anyone who attacks cherished beliefs and traditions. Today, it has finally come to mean something like, "incredibly nice fellow with closely-trimmed eyebrows."

Actually, I guess the middle definition of "iconoclast" – the one that talks about attacking cherished belief systems – is the one I would most hope to embody for ***JDJ***. (After all, I made up the one about the eyebrows.) Basically, if I were to set out to try to describe myself, I know the phrase "cynically skeptical" would figure prominently (as would the phrases "dashingly handsome" and "conditionally conscious").

Personally, I'm simply unwilling to take much of anything on faith, or because so-and-so says so. A couple of my heroes are James Randi and the late Richard Feynman, so you could say I'm not exactly in the mainstream of modern belief systems. So be it. With any luck, I'll do my eyebrows proud.

So, along those lines, I needed to come up with some cherished belief system to attack for this month's column. I couldn't just go after the low-hanging fruit though, like tarot readers or the Psychic Eyebrow Network or somesuch. And whatever I did come up with should be at least tangentially computer-related to fit into the context of this fine publication.

Well, a couple of weeks ago I just happened to be viewing the big screen – I had my remote control deftly clutched in the Random Button Access with Embedded Tobacco Harness position, and was flipping joyously amidst the tripe – when I suddenly experienced a catalytic inspiration for the choice of this month's "target" belief.

The image is of a young fellow, standing in a courtyard, positively raving about how he's listening to the radio outside...without *any* wires attached.

Naturally, I was a little confused. I mean, radios are wireless by definition, right? Didn't Guglielmo Marconi send wireless radio signals across the Atlantic Ocean almost a century ago? Didn't I snap my fingers to a wireless transmission of "The Immigrant Song" in my $100 1962 Studebaker Lark? I mean, what's the big deal?

Then the commentator went on to explain that this is Internet radio he's listening to on his little handheld device. It's streaming audio arriving over a 24Kbit/sec wireless Net connection (barely phone quality). I guess what struck me as funny was the amazement in the commentator's voice.

"Wow! I'm actually standing out of doors listening to this streaming Internet radio! Who could've imagined?" Of course, the whole kit and portable kaboodle – hardware and software – would set you back almost $1,000. (Hopefully, it might also do other useful things, like provide driving directions to that poorhouse in your future; he didn't say.)

Oh, I know this Net radio device is really just a "proof of concept" for the whole "wireless connectivity" belief system. One of the most prominent features at JavaOne was the ubiquity of connectivity, both on the show floor and among the attendees. Everywhere you looked, people had tiny phones that could start their cars, feed their pets, and ignore their e-mail.

I can't deny that this connectivity has its upside. But do I really want to *need* a cell phone? I'm not so sure.

Currently you're effectively antediluvian if you don't have one of these devices, the tinier the better, hanging from your belt or your ear or your spaghetti strap. I'm going to theorize that it won't be long before the lucky ones are those who don't have to be connected. Frankly, most of the time I don't want to be 10 dialed DTMF digits away from anyone in the world. In fact, there are times when I'd simply like to say, "Gone fishin."

How's that for radically iconoclastic?

AUTHOR BIO
Blair Wyman is a software engineer working for IBM in Rochester, Minnesota, home of the IBM iSeries.

blair@blairwyman.com

Java Dudes