ORACLE8i: THE FIRST INTERNET DB

JAVA™ DEVELOPER'S JOURNAL

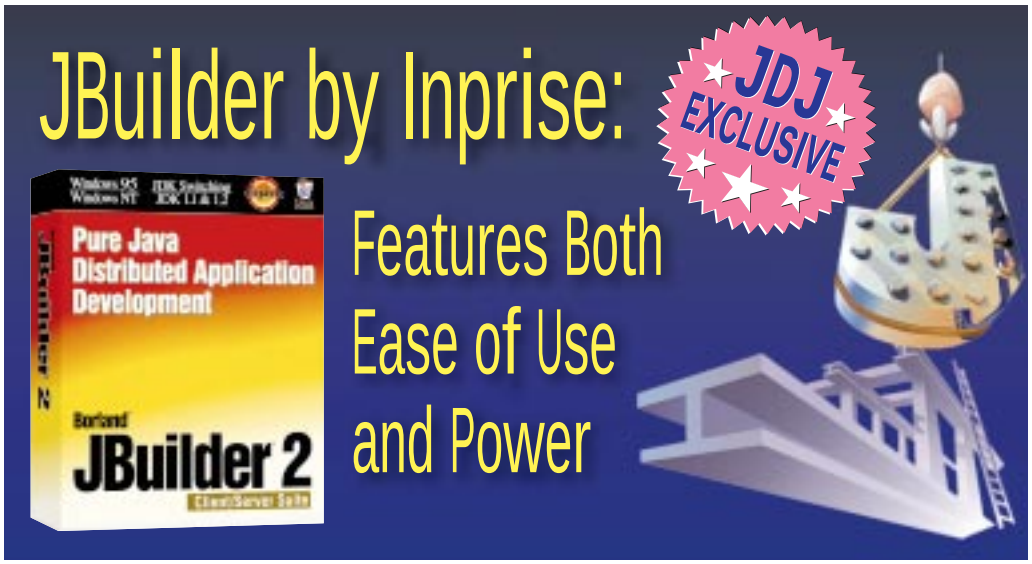
JavaDevelopersJournal.com

Volume: 3 Issue: 11, 1998

FREE CD
JBuilder 2
Referentia

RETAILERS PLEASE DISPLAY UNTIL JANUARY 31, 1998

SYS-CON PUBLICATIONS



EDITORIAL OFFICES

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

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

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

FROM THE EDITOR

Sean Rhody, Editor-in-Chief

The Perfect Beast

Build a better mousetrap and the world will build a better mouse. In the beginning we had a two-tiered architecture (I count mainframes as prehistory), and we could figure out how to do things with it. Unfortunately, one of the things we figured out was that we needed more than two tiers. Up came the concept of an application server and a Web server to accompany our ubiquitous database server.

I've had occasion recently to look at a number of currently available application servers. In general I like what I see, but as usual there isn't one single product that fulfills every need, although there should be. That's my opinion of course – but that's what this column is about.

The purpose of an application server is to provide a convenient integration point for business logic, reduce or remove business logic from client applications, minimize network traffic and provide an open interface to the business logic. No server is perfect, but here's my list of minimum features that make up my ideal starting point.

The first place I look is language support. I'm a big supporter of Java, but let's be honest: it's not the only language in use and it never will be. That's not just on the server side either. I know of many shops where Java in a browser is prohibited, and just as many who feel the same about ActiveX controls. The point is that an application server should support several languages and paradigms, from both a component standpoint inside the server and a client standpoint. To me the minimum should include Enterprise JavaBeans, regular Java classes, C/C++ code and ActiveX. I've found many application servers out there that will support Java, but only a relative handful that will support ActiveX. It's a big world, and many companies have ActiveX components or products that can become part of an application. I believe in building best-of-breed solutions, not just best-of-language.

Similarly, the methods provided for accessing the server should be as broad as possible. To my mind CORBA IDL, RMI and COM support need to be present. One vendor I know of goes so far as to support method calls as if they were stored procedures and the application server was in fact a database. In general, supporting CORBA and COM clients allows greater flexibility in client languages and deployment environments, while RMI expands the field further for us Java fanatics.

Transaction services should also be part of the package. Ideally, the server-side developer should be able to concentrate on writing code without worrying about database transactions. This is one area that shows the most variation. Almost all servers have some transaction support, but some provide their own. Some allow you to hook into existing products or specifications such as Tuxedo, DTC or XA, and some support the JTS standard. I'm on the fence about which is more appropriate. Pretty much any one of these approaches works, so the idea is to insist on some transaction service rather than coding your own. Some servers that support multiple component types don't allow transactions between types (for example, while some servers may use JTS but allow COM components, without some fancy footwork the COM component may not be able to take part in a transaction). If heterogeneous server code is a fact of life for you, make sure your server treats all components equally.

Right up there with transaction management is connection caching. A connection cache allows a small pool of database connections to service a larger number of user connections. For example, a five-user Oracle license might serve 25 clients simultaneously. The idea is that no database connection is used 100% of the time, so the connection cache can multiplex the connection for greater throughput.

Security services and network management are also features to look for. The typical approach is to apply a role to a component. This may not be granular enough if you need to apply security or data member to a particular function rather than at the component level. This is one area where no vendor really provides sufficient functionality. The ability to manage the server remotely, and for the server to participate in network management products such as HP Openview or some other SNMP-compatible product, is also a key point.

Finally, the server should support debugging. If you can't debug running code, your development time goes through the roof. Any vendor who says you can run the code first locally to see how it'll react is snowing you – you need to be able to debug inside the server.

So there's my minimum perfect beast. As soon as someone builds it, I'll let you know (and then I'll come up with a new set of features). If you'd like to help me with this, come visit me at the Java Internet Expo on December 8 in New York City. I'll be there with other members of the ***JDJ*** staff to meet with you, answer questions, hobnob with the big shots and (dare I say it) sign autographs. See you there.

About the Author

Sean Rhody is the editor-in-chief of **Java Developer's Journal**. *He is also a senior consultant with Computer Sciences Corporation where he specializes in application architecture, particularly distributed systems.He can be reached by e-mail at sean@sys-con.com.*

GUEST EDITORIAL

Samir Mehta

Are We There Yet?

After my family and I moved to Seattle, my friend Barry visited us with his wife, Mary, and their beautiful daughter, Julia. We hadn't visited Mt. Rainier so we thought it would be a good idea for all of us to drive there together. We weren't even half an hour into the trip when Julia asked, "Are we there yet?" That wasn't the only time she asked. The trip turned out to be excellent, but I didn't know then that I would soon be asking myself the same question for a different reason.

Barry is a technical writer turned developer at a reputable software company. During his visit, our discussion turned to the industry's adaptation of Java as a language and platform. He asked me if "Java was there yet" – if Java had industry support and wasn't just hype anymore. I was a little hesitant in answering that question, especially since Java had just celebrated its third birthday. So I decided to apply my trusted "dependency requirement algorithm" recursively to the problem at hand.

How does one determine if the industry has adopted a new standard? Look for adopters! My Internet search generated hits in excess of 4,092,640 for Java companies ranging from software giants like Microsoft, IBM, Sun, Oracle, Symantec and Computer Associates to minnows like 4thpass, WoodenChair and ObjectSoft. Well, I thought, these are the software companies and they're likely to adopt Java earlier. Upon further digging, when I found that corporations like Siemens, Xerox, Volvo and Citibank were using our product, SourceGuard, it was hard not to notice an emerging trend in the corporate development environments. Also, the fact that more than one third of the SourceGuard downloads were from outside the U.S. indicated that the adoption of Java was not just domestic but global.

In order to adopt Java, what do these corporations need? First and foremost, they need personnel with domain knowledge. My Internet search revealed Java-related job hits in excess of 6,008,332! Not a small number, even if you consider those that repeated, cross-referenced or were out of date. I came across sites solely advertising Java jobs (www.javajobs.com, www.javajobsite.com) to online recruiting services like www.dice.com. Almost all the printed and online magazines I came across advertised Java openings. Who caters to the needs of these personnel? From JavaSoft (www.javasoft.com) to IBM's JCentral (www.ibm.com/java), many thousands of dedicated and nondedicated sites provide users with up-to-date information on Java-related topics. Then there are Usenet newsgroups for discussing Java topics (comp.lang.java.*) and private newsgroup postings and forums like the one hosted by ***Java Developer's Journal (JDJ)*** online magazine (**www.JavaDevelopersJournal.com**). These resources are the tip of the iceberg. You just can't overlook printed and online dedicated magazines like ***JDJ*** and other nondedicated magazines. On top of these, Java books fill up shelves in major bookstores. In an online bookstore, Amazon.com (www.amazon.com), a simple search revealed some 400+ books on Java.

After completing my search for resources for personnel, I decided to check tools necessary for a productive environment. I found tools required for various stages of product development, ranging from requirement analysis, design, development, debugging, protection and installation to marketing. Searching for the right tool was a matter of visiting ***JDJ***'s online buyer's guide, JavaSoft's Java Reel and *Java Solutions Guide*, not to mention scores of online e-commerce stores.

If available resources are any indication of the adoption of Java, the answer is clear. This is without even touching the embedded market segment! Just as with everything else in life, there are issues that need to be addressed – Sun versus Microsoft, JFC versus WFC, Java standards body versus JavaSoft and many more.

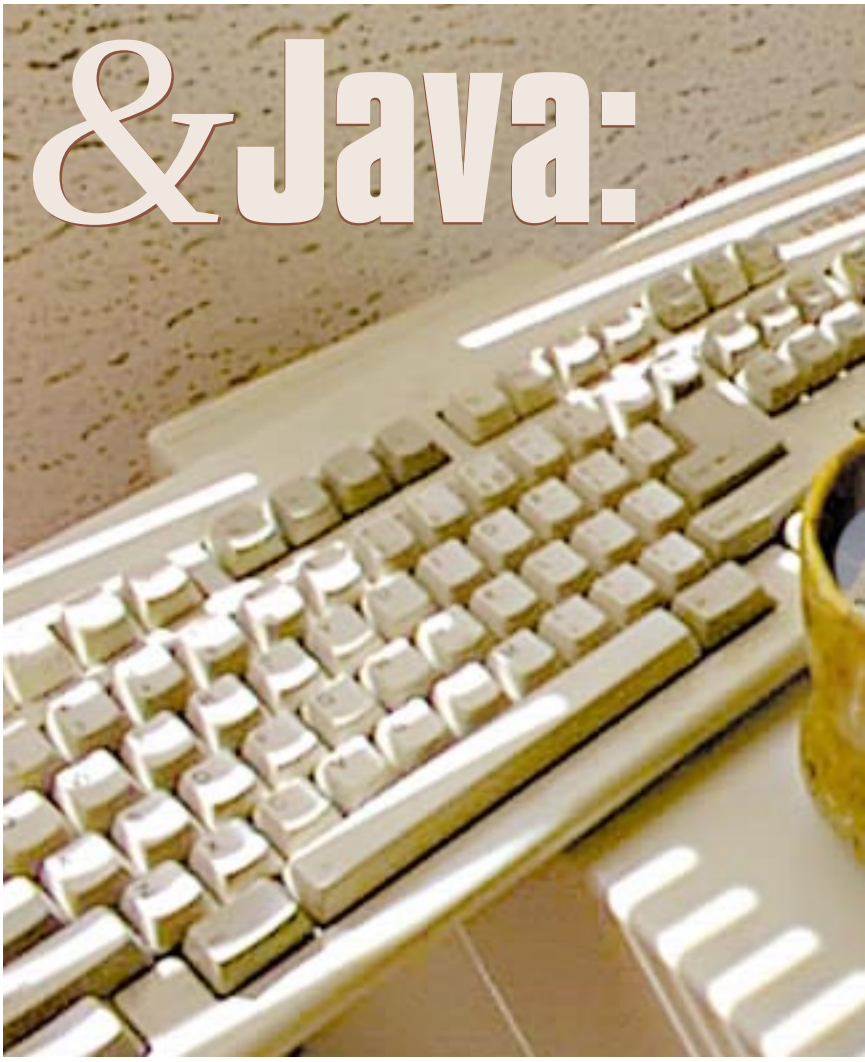
Well, at least I'm clear about my answer to Barry: "Yes, it looks like we're almost there."

About the Author

Samir Mehta is chief technical officer and cofounder of 4thpass LLC, a market leader in Java bytecode protection. He can be reached at samir@4thpass.com.

Oracle8i & Java: An Enterprise Java Platform

Build and deploy enterprise applications using open Internet standards



by **Thomas Kurian and Dave Rosenberg**

The Internet is rapidly evolving from a static, stateless, information-exchange medium to a dynamic transactional medium that offers new opportunities to change how we do business. To exploit these opportunities, leading-edge independent software vendors and corporate IT organizations are beginning to build and deploy enterprise applications using open Internet standards. They use standard Internet browsers and HTML as clients, model their business logic in Java and access these applications via open Internet communication protocols such as HTTP and IIOP. They deploy applications on a small number of professionally managed, highly scalable and high-performance application servers and database servers.

Building enterprise applications on the Internet model has been challenging, however, requiring application developers to learn an enormous number of tools, and different types of servers and middleware. Further, the infrastructure platforms available have failed to offer the performance, scalability or reliability required by enterprise applications. Java servers in particular have faced three important limitations. First, the Java language is challenging, requiring automatic storage management, language-level multithreading and loosely coupled dynamic loading that limit scalability and performance. Second, the first generation of Java VMs have naturally focused on the needs of single-user systems; for example, good, lightweight performance for dynamically downloaded applets, and GUI-driven execution rather than the enormous throughput, performance (for long-running, preinstalled applications) and reliability required for mission-critical, transaction-oriented business applications. Third, Java VMs have required developers to write multithreaded servers using sophisticated multiprogramming concepts rather than providing a robust, multithreaded server platform that simply runs business applications scalably.

Oracle8i was designed to dramatically simplify how Internet applications are developed, deployed and managed. There are three fundamental components to this strategy.

- At the heart of Oracle8i is a fast and highly scalable Java Virtual Machine that is seamlessly integrated with the database kernel and provides an efficient execution platform for Java programs. It was designed specifically to address the three challenges described above – performance, scalability and reliability.
- Oracle8i provides users with a variety of industry-standard programmatic interfaces with which to build applications in Java. These include support for Enterprise JavaBeans, CORBA Services in Java and Java stored procedures, an emerging standard supported by virtually all the major database vendors. Java applications in turn access SQL via standard interfaces – JDBC drivers or SQLJ, a standard way to embed SQL statements in Java programs.
- Oracle offers JDeveloper, a Java development tool that provides a number of facilities supporting SQLJ, Java stored procedures, Enterprise JavaBeans and CORBA services

Key Architectural Features of Oracle's Java VM

Figure 1 provides an architectural illustration of Oracle8i's JVM. Three primary architectural components make it an efficient server platform.

- An advanced server-oriented memory manager to improve the scalability of Java programs
- A high-performance native compiler to speed up the execution of Java programs
- A generalized protocol framework to allow Java programs to be accessed directly from a variety of Internet clients

Server-Oriented Memory Management System

The Oracle database has for many years supported a highly scalable server called the MultiThreaded Server (see Figure 2) within which we have integrated Oracle's JVM. The JVM includes a unique, high-performance memory manager that conforms to and is tuned to the multithreaded server's shared memory model while remaining completely transparent to the Java programmer. It allocates and frees memory in chunks called object memories. There are three different types of object memories with different lifetimes and degrees of sharing across users:

- *Shared memory:* The JVM puts immutable Java objects (such as metadata, bytecode vectors and constant pool data) into shared memory initialized once and shared across users.
- *Session Memory:* It places Java session state (the transitive closure of each concurrent user's static variables over their references) into per-client shared memory, giving users the illusion of their own "virtual machine."
- *Call Memory:* Finally, it uses per-call memory as the allocation space of a modern-generation, scavenging garbage collector.

The net benefit of these sophisticated shared-memory mechanisms is to reduce per-user memory requirements for typical stateful Java sessions to between 80 and 150 KB, allowing Oracle8i to support tens of thousands of concurrent users.

Native Compilation

Contrary to early impressions, Java is not inherently slow to execute, but is amenable to a wide range of compilation strategies. Oracle8i's JVM includes a native compiler tuned to the needs of server applications. It translates Java binaries to a portable subset of ANSI C source, which is in turn dynamically compiled to platform-specific binary code by a target C compiler. The resulting exe-

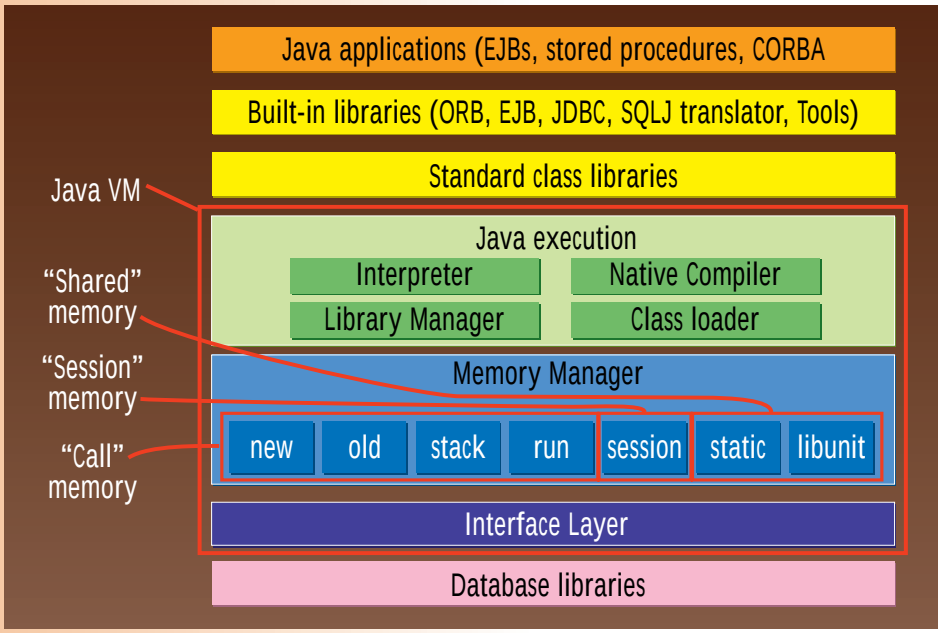


Figure 1: Oracle's JVM architecture

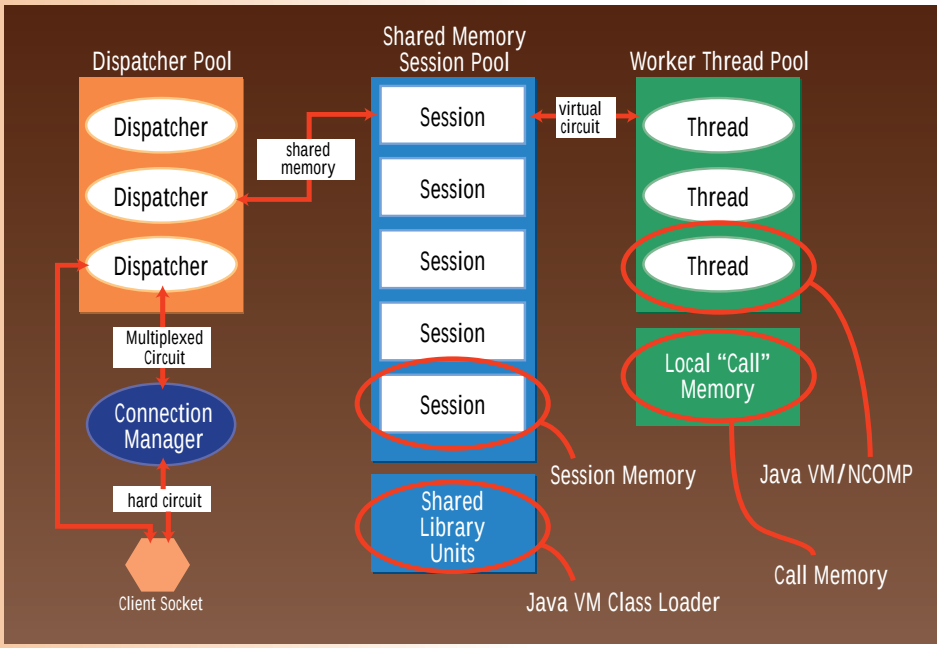


Figure 2: Multithreaded server architecture

cutable is dynamically linked directly into the server as a dynamic-link library. We adopted this approach for two reasons: first, server code tends to be longer lived and more throughput-critical than client code, allowing us to invest more resources upfront to produce more efficient code than JIT compilers, which must amortize the cost of compilation over a much shorter duty cycle; and second, it allows us to offer high performance uniformly on over 60 different hardware and operating system platforms.

The net benefit of native compilation? In early tests the native compiler has demonstrated improved Java execution performance from 15 to 40 times faster than interpreted code.

Generalized Protocol Framework

To allow a variety of standard Internet clients such as browsers, Internet mail clients, embedded ORBs and middle-tier application servers to access Java programs scalably in the server, we have generalized the database's listener and dispatcher to provide general-purpose protocol handling for arbitrary, user-defined protocols. Oracle8i uses this infrastructure to provide native support for HTTP, IIOP, Internet Mail protocols (IMAP4, SMTP, POP3) and other protocols including FTP.

Programming Oracle8i's JVM

Oracle8i's VM is a general-purpose JVM; any Java application can be run on it. It supports three different programming models: Java stored procedures, Enterprise JavaBeans and CORBA Services.

Java Stored Procedures

Java programs can be stored and executed in the Oracle database as Java stored procedures, which allow users to program the database by adding business rules in Java to extend SQL. As an industry standard, Java stored procedures are portable across databases from many different vendors. Such procedures use JDBC or SQLJ to access data. Because they execute in the database server, SQL access is much faster than when the data needs to be retrieved from the server to a JVM on another machine. Java stored programs run in a variety of contexts:

- User-defined functions and stored procedures are called within SQL queries, allowing the stored procedure programmer to directly extend SQL with Java.
- Triggers are stored procedures that are tied to a particular table or view and execute when that table or view is changed. All five types of Oracle triggers can be implemented in Java.
- Object-relational methods allow users to add behavior to SQL object types in Java. Java stored programs can be invoked from any database client including Java clients such as JDBC and SQLJ, 4GL tools such as Developer/2000 and PowerBuilder, and C/C++ clients via ODBC or the Oracle call interface.

Enterprise JavaBeans (EJBs)

EJBs give Java application programmers a convenient and highly productive component model for server-side business logic, facilitating code reuse and multitier application development. Oracle8i provides a highly scalable and high-performance execution environment for EJBs that complies with the EJB 1.0 specification. It supplies a number of EJB services including a Java Transaction Service (JTS) API via the embedded JDBC driver, which has been extended to support JTS-visible transactions. It exposes a Java Naming and Directory Interface (JNDI) to any industry standard Lightweight Directory Access Protocol (LDAP)-enabled directory service. EJB components in the server can be placed in any standard directory and accessed via JNDI. Additionally, it provides a stringent security framework using Internet-standard security mechanisms such as SSL over IIOP for encryption, coupled with traditional database authentication and multiple layers of access control.

CORBA Servers

Distributed systems developers can use the infrastructure used for EJBs to deploy CORBA servers implemented in

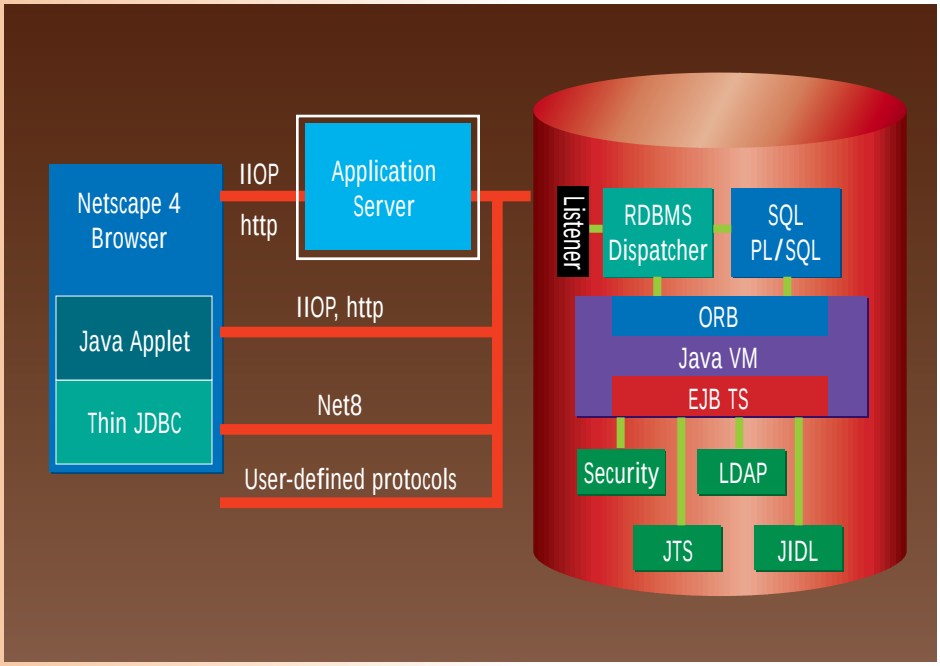


Figure 3: Using Enterprise JavaBeans and CORBA

Java. The JVM provides a CORBA Object Transaction Service (OTS) and also exposes a standard COSNaming interface. Oracle8i provides an object adapter that serves as a registry of CORBA objects published in the RDBMS, and helps locate and load CORBA objects upon initial activation by CORBA clients. It also provides a number of features that make it easy for Java programmers to develop CORBA services, including Caffeine, a direct Java-to-IIOP mapping that eliminates the need for IDL definition, support for objects by value and extensible structs, and a number of tools that simplify application development. Figure 3 illustrates how Enterprise Java Beans and CORBA services can be used in Oracle8i.

Summary

Oracle8i represents a revolutionary new breed of product: a server platform targeted specifically to simplify how you build, deploy and manage Internet applications. It combines enterprise-scale support for SQL data (the shape of the Internet's data) and Java programs (the language of the Internet's behavior) in a highly integrated, robust, easy-to-use package. All of its components – the protocol framework, the JVM and the data-management facilities – are configured to run out of the box and be managed with a single tool. Oracle8i promises dramatic improvement in how quickly and productively Internet applications can be developed and deployed.

About the Author

Thomas Kurian is director of Internet computing at Oracle Corporation and is responsible for formulating Oracle's database strategy in the Internet and e-commerce marketplaces. Prior to Oracle, Thomas worked at McKinsey and Company, an international consulting firm. You can e-mail him at tkurian@us.oracle.com.

Dave Rosenberg is senior director, Java products group, Oracle Corporation, and heads the development team responsible for the Java VM in Oracle8i. Prior to Oracle, Dave led efforts to build object-oriented databases at Object Design Inc. and knowledge management systems at ISX Corporation. You can e-mail him at darosenb@us.oracle.com.

tkurian@us.oracle.com darosenb@us.oracle.com

JSplash & JTips

Add a splash screen to your widget collection

by **Claude Duguay**

This month we'll build a pair of high-level widgets for application development. JSplash is a simple splash-screen window that displays a centered image for a time-out period – until the user presses a key or clicks the mouse. JTips automatically presents the user with a cycling set of tips at application startup. The tips are loaded from a simple text file, and JTips provides options so the user can either disable it or move through the list interactively.

The JSplash Window

Listing 1 shows the code required to present a splash-screen image to the user. The JSplash window is a fairly simple object, but it has to handle a number of circumstances correctly. Here's a short list of requirements:

- If the user does nothing, we should time-out after a given period.
- If the user presses a key or clicks the mouse, we dismiss the window.
- Startup conditions should be handled as automatically as possible.
- Programmers should be able to display the splash screen anytime.
- It should be possible to block until the window gets dismissed.

Figure 1 shows an example of JSplash in action.

If you look at the source code, you'll see that the image handling is done through the ImageIcon class and that the JLabel component is used to frame the picture. We use a simple BevelBorder to make sure the image is visually in the foreground and adjust the JWindow bounds accordingly. The constructor first loads the image file, then calculates the position and size required to center the window with setBounds.

Once the image is loaded and the window is properly positioned, we register the JSplash class as a KeyListener and MouseListener, registering the parent as a MouseListener as well. Then we create a JFC Timer, which has the specified time-out in milliseconds. The Timer, which never repeats, will send an ActionEvent when the time elapses, so we register to receive this event as an ActionListener.

With the exception of the block method, the rest of the code merely implements the listener interfaces we registered for. We capture each of the relevant events and dismiss the window by making it invisible before calling the dispose method.

The block method allows you to wait until the splash screen is dismissed before continuing (to avoid sequence problems). The calling thread will block until the splash screen disappears.

Listing 2 shows the JSplashTest class, which demonstrates an important trick you'll have to apply to properly capture keyboard events. First we create an arbitrary application frame and then the JSplash window with the frame reference, image file name and time-out period as constructor arguments. After calling the JFrame show method, we request the focus again for the JSplash object. If you don't do this, pressing a key may trigger something in the main application frame without dismissing the splash screen first – typically not what you really want to occur.

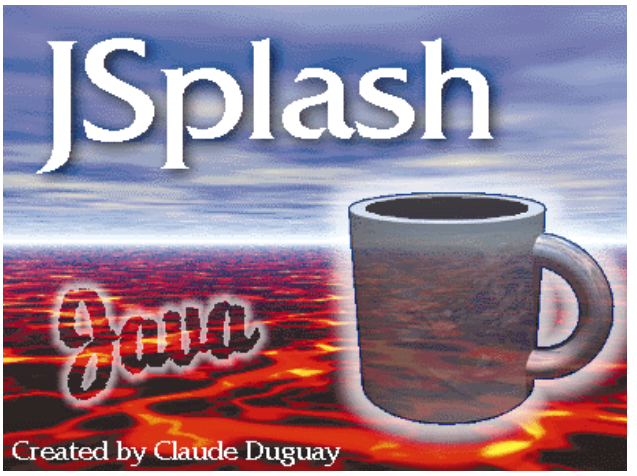


Figure 1: JSplash example window

The JTips Window

The JTips requirements are fairly simple. Show users a single text tip and allow them to move forward, cycling each of the tips in turn, and starting at the first one when we reach the end. We also permit them to dismiss the window and turn off the automatic tips at application startup. Figure 2 shows the visual result using an example tip file.

Listing 3 shows the JTips source code. The constructor first sets default dimensions and centers the window on the screen. We store the filename and properties arguments, allocate a vector to store the tips and then call readTipsFile to handle the file content. The rest of the constructor sets up the many panels that create the visual design. Figure 3 shows how these panels are nested to achieve the desired effect.

We use the panels to control positioning and the effects of resizing. The shaded left icon panel, title and button panel always surround the tips panel and get repositioned accordingly if the window size changes. The navigation panel keeps the buttons grouped together to the bottom right of the window. The show checkbox is always at the bottom left.

Most of the remaining code in JTips is dedicated to handling the tip text. We use the tips Vector to store the sequence of tips read from the file, and the Properties object to store the current position in the list. This design choice makes using configuration files as easy as possible. Normally, you'd save the application-related properties in a file and load it when the program starts. Permitting this value to be passed into JTips keeps the coupling to a minimum. JTips stores two properties called "tips.index" and "tips.show".

The readTipsFile method opens the tips file and reads each line before closing it. Each line is preprocessed by replaceParagraphMarkers before being put into the tips vector. The replacement code

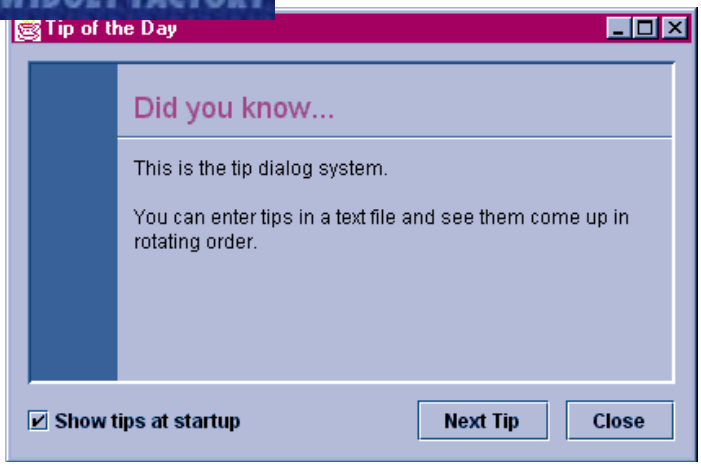


Figure 2: JTips example window

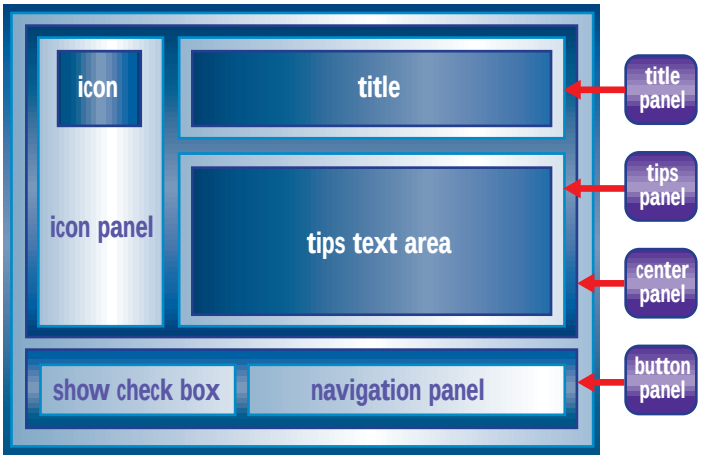


Figure 3: Nested panels in JTips

recognizes the "\p" and replaces it with two new line ("\n") characters. Since each tip has to be on a single line, this allows you to create multiparagraph tips when needed.

A number of utility methods are also implemented. The getNext and setNext methods are accessors for the index property. The getShow and setShow methods do the same for the show property. The increment method cycles to the next index position and saves the new value. The nextTip method actually takes the text from the given index position and sets it to be displayed, then calls increment to move the index position forward.

Finally, we catch button events with the actionPerformed method, which implements the ActionListener interface. The three buttons toggle the show state, go to the next tip or cancel (making the window invisible). You can redisplay the window by using the show or setVisible methods. The startup method does this automatically if the show property is on. It can be used in your main clause to make this virtually transparent.

Listing 4 shows the source code for the EdgeBorder class. The code is long but simple and implements the Border interface. We use it in JTips to draw the line at the bottom of the title panel. EdgeBorder draws an etched line to the north, south, east or west, with the etching either raised or lowered. Although we need only the lowered north variation, it doesn't make sense to implement a border without a proper design.

The EdgeBorder class has to implement the getBorderInsets method as part of the Border interface. We return a two-pixel inset on the selected edge. We also implement isBorderOpaque to always return true, and the paintBorder method to draw the actual border. The code uses case statements to decide which lines to draw on the selected side. Each of the two lines is shaded based on the background color of the component in which the border is used.

Listing 5 shows the code for TipTextArea, which subclasses JTextArea so as to control certain key properties more effectively. We set the font, an empty border to control the margins, make the editable property false, and set the word wrap "on" and "word-based." We also return false for the isFocusTraversable method so that the focus is never set on this control.

> ***"The splash screen adds an element of professionalism to any application."***

The ApplicationTest Harness

Listing 6 shows code for the ApplicationTest class. Most of the work is done intentionally in the main method to demonstrate how you'd set things up in a typical application. The constructor simply sets up the BackgroundPanel from Listing 7, which merely draws alternating blue and dark-blue stripes in a panel so you can see some contrast when you run it.

The ApplicationTest main clause creates an instance of itself and of the JSplash widget and then centers the main frame on the screen before showing it. We call requestFocus on the splash window to make sure we can capture keystrokes. We then create a new instance of JTips and call the JSplash block method to wait until the splash screen disappears before showing the tips windows. That's all there is to it.

In a real application you'll probably want to add a pair of "Help" menu items to call up JSplash under "About…", and to redisplay the JTips window under "Tip of the Day...". We use the ellipsis (...) convention to tell the user they'll be seeing a window when they select those menu entries.

Summary

This installment of the Widget Factory provides a pair of useful components you can add to your growing widget collection. The splash screen adds an element of professionalism to any application, offers an opportunity for custom branding and facilitates a strength of identity for the product with virtually no programming effort.

The JTips control is especially important in applications that aren't so obvious when the user sees the interface for the first time. If your interface is difficult to use, you should consider a redesign; if it's easy to use but nonobvious at first, a simple Tip of the Day is precisely what you need to help the user overcome those small, early barriers as painlessly as possible.

CODE LISTING

The complete code listing for this article can be located at **www.JavaDevelopersJournal.com**

About the Author

Claude Duguay has been programming since 1980. In 1988 he founded LogiCraft Corporation, and currently leads the development team at Atrieva Corp. You can contact him with questions and comments at claude@atrieva.com.

claude@atrieva.com

The Java Programming Language Trade-offs

Java is doing a pretty good job of defining its turf, but it doesn't provide a solution for every problem

by **Ajit Sagar**

So I get to the office in the morning and see Mr. Job Prospect's résumé lying on my desk. That gives me about 20 minutes to think of interview questions I'd like to ask him. A quick scan of the résumé reveals that he's done some serious work in Java that includes programming with JFC, JavaBeans, Java threads, Java Applets – the works. As usual, I decide to play devil's advocate. That always throws them for a loop.

After the regular stuff, which Job passes with flying colors, I ask, "What's so great about Java?" He gives me the look I've received from several candidates – a look that says, "What a silly question! The answer is obvious. Java is the coolest, OO-pumped, distributed, reusable, superduper, kick-ass language there is." The next question really has Job questioning my sanity: "Why not use C++?"

Java is a very productive language that's revolutionized business computing. It's a clean language that fosters good design practices and provides inherent support for object-oriented design. However, it's neither a panacea for all computing ailments nor the utopia of the programming world. Choosing Java over other programming languages comes with its own share of compromises and trade-offs. A common misconception is that Java is here to replace C++ (and every other language) and is going to be the only language in the marketplace. Java is certainly making an unprecedented impact in the computing arena. But a large part of the computing problems that the Java programming language addresses are ones that have appeared with new computing paradigms. Most programming languages that manage to survive more than a couple of years in the computing marketplace have merits and unique strengths that make them ideal for their turf. Java is in the process of defining its turf and is doing a pretty good job. However, it doesn't provide a solution for every problem in every domain.

In this article I'd like to discuss a couple of features and idiosyncrasies of Java and the costs of their usage. We'll look at the lack of multiple inheritance in Java, garbage collection, references

and an interesting peculiarity of Java's method scoping.

To Multiply Inheritance or Not?

One of the main language features that the designers of the Java language decided to drop off the plate was multiple inheritance. Inheritance in the context of object orientation is an "is–a" relationship. Multiple inheritance implies an "is–also–a" relationship. When a class inherits behavior and state from more than one class, we run into multiple inheritance. The real problem in multiple inheritance arises when the derived class inherits the same state (or member variables) from two different superclasses. This is known as the infamous DDD (Deadly Diamond of Death) and is illustrated in Figure 1. The problem surfaces when the derived class refers to the variable foo. Does it refer to Super1.foo or Super2.foo? Multiple inheritance increases the complexity of the code as well as the compiler.

Java avoids this problem by enforcing a policy so that a class can inherit state and implementation from only one superclass. This doesn't preclude the class from inheriting pure behavior from multiple interfaces. The mechanism of inheriting state and/or behavior from a superclass is called *class inheritance*. The mechanism of inheriting pure behavior from a superclass is called *interface inheritance*. Java provides constructs to distinctly specify one or the other – the corresponding keywords in the language are *interface* (pure behavior) and *class* (state and/or behavior). Since the superclass methods have no implementation in the declaring class, the implementations for the methods defined in the superclasses have to be provided in the derived class. Hence a class in Java can simultaneously inherit from several interfaces and one single class. This is shown in the following class declaration:

```
public class Derived extends Super implements Interface1, Interface2
```

The class Derived inherits state and behavior from Super and has to provide implementations for the methods declared in Interface1 and Interface2. In most programming scenarios this stipulation leads to well-designed programs. However, there are cases in which it would be convenient to inherit state and implementation from more than one class. Consider a situation in which you have to write a small class (with maybe five additional methods) that extends the functionality of two classes – Super1 and Super2. This situation is common if you're using third-party class libraries or legacy code in which the implementation is already provided. Since you can only inherit the state and implementation from one superclass, you'll have to do the following to reuse the functionality of the other two classes:

- Write interfaces that define the method signatures of the superclasses. Let's call them Interface1 and Interface2. Chances are, these interfaces will be available to you with the third-party or legacy code. However, you might just get the compiled class itself.
- Extend Derived from Super1 and implement the interface of Super2.
- Define a reference to an object of type Super2 in Derived.
- Provide "wrapper" methods to delegate every method in Interface2 via the instance variable.

This mechanism is illustrated in Listing 1. The first two class implementations show Super1 and Super2. It's assumed that these implementations are already available and need to be extended by Derived. The next section of the listing shows the interface for Super2. Finally, the implementation of Derived is shown. Derived extends Super1 and implements Interface2 for Super2. Note that Derived doesn't have to provide wrapper methods for Super1 as it extends the implementation. However, for every method in Super2, Derived has to provide a wrapper method that delegates the call to the local reference to an object of type Super2.

Now consider that Super2 has 30 methods. To make matters worse, suppose that Derived needs to extend the functionality of another superclass with another 30 methods. Derived will have to provide 60 methods that do nothing but provide wrappers for existing implementations. This produces a "class bloat." You'll also notice a "class creep." Note that you needed to declare an extra interface for Super2. This may seem a small price to pay, but consider that in this scenario, to inherit a class, you have to add an interface. If you scale this to a large project, you end up with a large number of class/interface declarations.

If we had multiple inheritance in this case, all we'd need to do for Derived would be something like this:

```
public class derived extends Super1, Super2
{
   // Only code specific to Derived class goes here
}
```

Using single versus multiple inheritance is illustrated in Figure 2. In the figure the black arrows represent inheritance from a superclass to the derived class; the red arrows indicate interface implementations; the green lines, delegation.

It would be nice if the language had a mechanism for directing the compiler that indicated to the compiler that a class is a delegate for another class. This would at least prevent the need for derived classes to provide all the wrapper methods.

Garbage Collection

One of the major productivity gains in using Java for development came about because Java doesn't require the programmer to manage memory deallocations. In a pure Java program there's no such thing as a memory leak. Memory is allocated at instantiation; however, it's automatically released by Java's garbage collector. Freeing an object in Java is a simple task of setting its reference to null. The garbage collector frees the memory when it next collects garbage.

This is a nice scheme that makes programming easier and less prone to errors. The caveat is that the programmer has no control

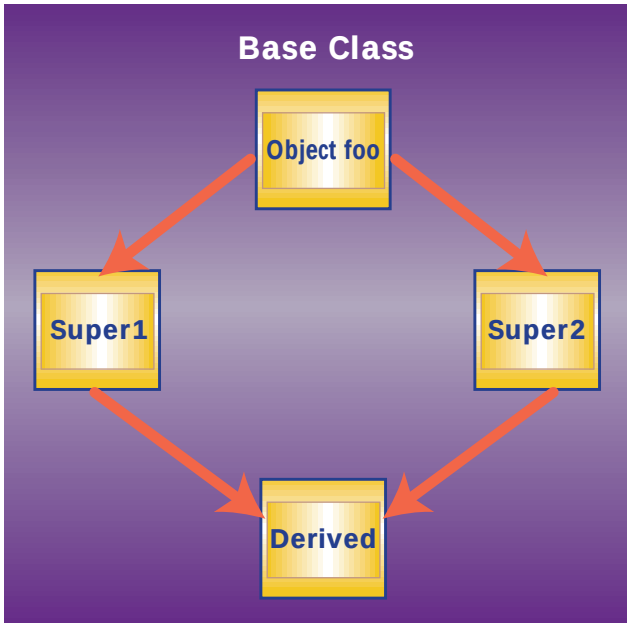


Figure 1: The Deadly Diamond of Death

over when the garbage collector runs. It runs at the discretion of the Java Virtual Machine. Typically, it runs periodically and "cleans up garbage." Different VMs support different strategies for garbage collection. The bottom line is that memory is reclaimed when there's no longer any references to it.

Garbage collection works well in most programming situations. However, lack of control over memory deallocation sometimes presents interesting problems. The two main ones are in memory-constrained and real-time applications. Since the programmer has no control over exactly when memory is released, applications that are severely memory-constrained will have to rely on the garbage collector to free up memory fast enough so that no significant performance penalties have to be paid. This is typically not a problem as garbage collection algorithms have been around for a while now and are usually very efficient.

In real-time applications the problem has to do with the system resources that the garbage collector is going to consume when it runs. In these applications timing is critical, and the slowdown caused by the garbage collector may not be acceptable. Since there's no way to predict precisely when the memory is going to be freed, i.e., when the garbage collector is going to be activated, finding a workaround is a daunting task. The Java Runtime class provides a method gc() to facilitate garbage collection. A call to gc() may be made as follows:

```
// Tell the garbage collector to free up memory
System.gc();
```

The purpose of this method is often misunderstood. Calling gc() doesn't deallocate memory. It's merely a hint to the VM to run the garbage collector as soon as it can. When the garbage collector actually runs depends on the runtime environment and the implementation of the garbage collector.

References

Garbage collection is based on determining when an object in the runtime process is no longer in use. In a Java application all handles to memory allocated by the application are via references to the corresponding objects. The task of the garbage collector is to identify the references to objects no longer in use and to free the memory allocated to those objects. A runtime Java application contains all the objects created during program execution, which are stored in a root set of references. As long as a reference is in the root set, the object is reachable by the program. Once the object is freed (either by setting the reference to null or by the object's finalize() method), it's ready for garbage collection (as long as no other references still point to the object). At this time the object is unreachable. Unreachable objects are ready for garbage collection.

JDK 1.2 introduces a new API that supports a finer grain of control for Java program's interaction with the garbage collector. This API, called Reference Object API, is in the package java.lang.ref. The reference object encapsulates a regular reference to a Java object (known as a *referent*). The Reference Object API allows developers to define degrees of "reachability." Besides defining an object as reachable or unreachable, the API also defines the following reference types (in order of reachability): Soft reference, Weak reference, Phantom reference.

A detailed discussion on the Reference Object API is beyond the scope of this article. The impact on garbage collection is that the weaker the reference, the more the incentive for the garbage collector to free its memory. Creating appropriate references using the Reference Object API gives the programmer more control over what memory is freed by the garbage collector. A good source for more information on Java references is http://java.sun.com/docs/books/tutorial/refobjs/index.html.

An Observation About Method Scoping

I'd like to wind up this discussion with an interesting aspect of Java method scoping that one of my colleagues accidentally discovered. In JDK 1.01 Java supported a keyword *private protected*. Methods declared with the private protected qualifier were visible only to their inherited classes. This keyword disappeared in JDK 1.1.x. Currently the following scopes exist for a method:

- *Private* – A private method is visible only to methods in the declaring class (or in classes inside the declaring class).
- *Package* – A package scope is the default and is in effect if none of the visibility keywords (private protected or public) qualifies the method. A method with

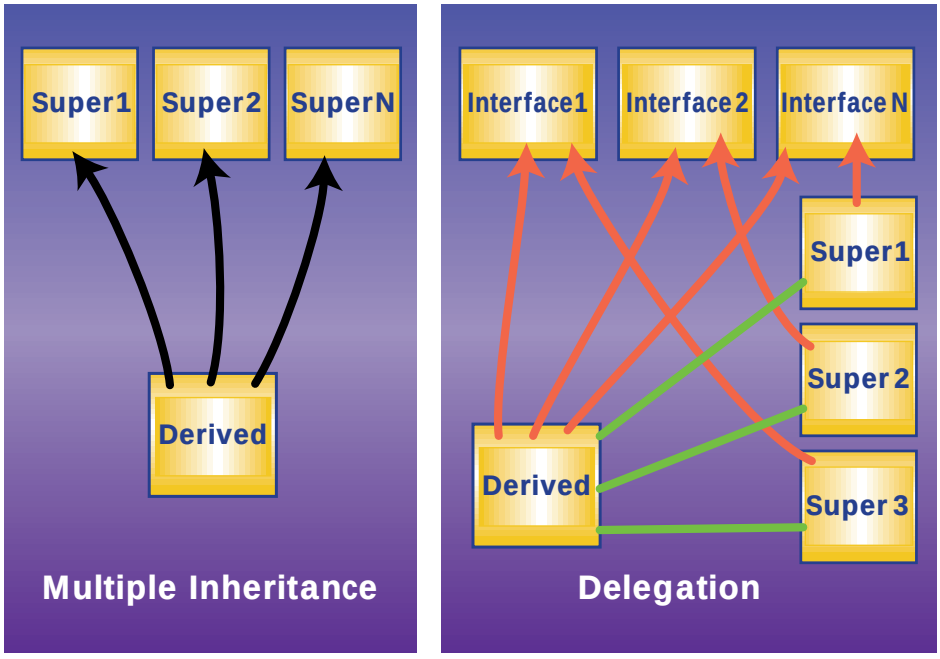


Figure 2: Multiple inheritance versus delegation

package scope is visible to methods in the declaring class and all classes in the package. Note that it's not visible to inherited classes that are not in the package.

- *Protected* – A private method is visible to methods in the declaring class, all subclasses of that class and in all classes in the package.
- *Public* – A public method is visible to methods in all classes in the application.

None of these method-visibility qualifiers limit the visibility to classes' direct inheritance hierarchy. For example, consider the following inheritance hierarchy:

```
public abstract class Base
{
  // Declarations and code

  // Declaration of foo
  <Scope Modifier> void foo();

   // more code
}

public class Derived extends Base
{
  // Declarations and code

  void foo()
  {
      // implementation
  }

  // more code
}
```

The Scope Modifier could be one of the keywords – public, protected – or neither (which implies default package scope). None of these limit the visibility of the method foo() to just these two classes. If the qualifier is protected or default (no qualifier), foo() is still visible to at least other classes in the package. What this means is that to limit the visibility to direct inheritance hierarchies, you'd have to package each inheritance hierarchy separately, i.e., Base and Derived would have to be in a separate package and the method qualifier used would be protected.

There is a way to limit the scope. If I change the code excerpt above to the following, the scope will be limited to direct hierarchy:

```
public abstract class Base
{
  // Declarations and code

  abstract private void foo();

   // more code
}

public class Derived extends Base
{
  // Declarations and code

  private void foo()
  {
     // Implementation
   }

   // more code
}
```

Declaring the method private makes it invisible to all classes except Base. Declaring it abstract, however, requires that a derived class provide an implementation of the method. The method is callable only from Base and Derived. The scoping rules are contradictory. A private modifier limits the scope to the declaring class only. Subclasses are excluded from the scope of the method. However, the abstract modifier apparently negates the stipulation.

About the Author

Ajit Sagar is a member of the technical staff at i2 Technologies in Dallas, Texas. A Java certified programmer with eight years of programming experience, including two in Java, he holds a BS in electrical engineering from BITS Pilani, India, and an MS in computer science from Mississippi State University. Ajit can be reached at Ajit_Sagar@i2.com.

Ajit_Sagar@i2.com

LISTING 1: Working Around the Lack of Multiple Inheritance in Java.

```
// Super1 implementation. This is already available.
public class Super1
{
   void method1()
   {
      // Implementation code here
   }

   // Other methods
}

// Super class 2 implementation. This is already available.
public class Super2
{
   void method2()
   {
      // Implementation code here
   }

   // Other methods

}

// Super class 2 interface. You have to write this.
public interface Interface2
{
  void method2();

  // Other methods declarations
}

// Derived class implementation.
public Derived extends Super1 implements Interface2
{
    // variable to store instance of Super2
    Super2 super2 = new Super2();

  // wrapper method for method2
   void method2()
  {
    super2.method2();
  }

// wrapper methods for all other Super2 methods

}
```

CODE LISTING

The complete code listing for this article can be located at **www.JavaDevelopersJournal.com**

Hmmmmm...What If?

An unsettling question

by **Alan Williamson**

It's that time of year again – the time when we all pretend to get along with one another for a few weeks. It's the time for families to come out of the woodwork, for getting out that knitted pullover from the Auntie whose name you can never quite remember. I can't wait until the New Year comes round again so we can go back to disliking people. Bah humbug!

Season's greetings to one and all. Actually, contrary to popular belief, I love this time of year. Over here in Scotland we generally get a white Christmas, and it's been known for me to get all "Bing" and burst out in floods of unrecognizable notes – which is never a bad thing!

Before We Get to What If...

It's Alan Williamson and here we are for another column of "Straight Talking." This has been an interesting month for me, one that has found me in all four corners of the world. My regulars will know that I usually take a characteristic and focus an article on it, contrasting it to our world of Java. Well, this month is going to be no different. I think we'll plump for dreaming. The ability to play "What if?" games. You know the sort: "What if I won a million dollars?" or "What if I was the best Java programmer in the world?" (We all think we're the best, so maybe that wasn't a good example.)

Anyway, this month my business took me back to Tokyo, and then over to Sydney, Silicon Valley, Boston and then finally New York. This was only my second time in Silicon Valley; the first time was when I was at JavaOne last year. But back then I never got a chance to look around the place – well, that's not exactly true!

I'm not sure how many of you have ever visited Silicon Valley. I suspect the majority of you have. Therefore you know what it is, and, more important, what it is not. I've been in this industry for many years, and during that time I've had this picture of Silicon Valley – a picture, I discovered, that was shared by many who've also never been there. I assumed Silicon Valley was one big industrial park set in acres of lush green gardens with signposts pointing to all the big players in the computing industry. Was I in for a shock!

The reason I missed it the first time around was that I was actually looking for such a place. I think I must have gone to every shopping mall trying to find the infamous "Silicon Valley." Time beat me then, so this time I got me a proper tour guide. James Davidson from Sun kindly lent me his know-how and drove me around Silicon Valley, pointing out all the major headquarters of the companies that shape our industry.

James showed me the offices of Sun, Apple, IBM, Netscape, Oracle and HP, to name some of the big boys. Each building, or set of buildings in some cases, was very impressive – and breathtaking when you think that what's potentially happening inside those buildings will affect every one of us and how we work. We have companies working on hardware, browsers, networks, databases and, most important, Java – all within a few miles of each other.

The Truth About Silicon Valley

For those of you who shared my view of Silicon Valley, let me tell you it's not like that at all. It's not one big industrial/business park. In fact, Silicon Valley is the name given to a region that technically doesn't exist. No map displays it. No signposts point to it. It's a collection of towns that among them house the most important and influential companies in the world.

I went to Boston next to drop in on one of the students we sponsor at Worcester Polytechnic Institute. I recounted the vision of Silicon Valley to Shahzad, and he agreed with me that his vision was somewhat aligned with mine. Then we got talking, and it's here that I'd like to bring you in.

Our discussions took us many a place, most of it dreaming, thinking about exciting innovations coming from that wee corner of California. Then it took on a sinister tone. We got worried. Here's where I posed the question, "What if we woke up one morning and Silicon Valley was no longer there?" The two of us just stared at each other for a moment, each of us mentally calculating the impact that "What if" would have on the world as a whole.

Now, we know the chances of this actually happening are small. We all know that the entire state of California is on a famous fault line, the San Andreas Fault. One of the biggest misconceptions about this particular fault line is that one day California will fall into the sea. Before writing this article, I did some research into it and came up with some interesting facts and figures. For example, California will not fall into the sea, but instead it is moving up toward Alaska and Canada at a rate of around 35 mm a year. So Bill Gates is going to have Scott McNealy as a neighbor after all!

Second, in the last 10 years California has experienced around 20,000 tremors. Granted, most of them have been small, but considering that every point in California is only 30 miles away from a major earthquake zone, you have to wonder whether Mother Nature engineered Silicon Valley to be where it is as part of her global natural selection program.

On another note, in Cold War times it was rumored that most of Russia's missiles were aimed at California. This would have wiped out most of the entertainment and computing industry. Couple that with the earthquakes, and suddenly old California doesn't look so appealing.

But back to the question I posed. What if we woke up one morning and suddenly the headquarters of Sun, Netscape, Apple and Oracle were no longer there? Everything was lost. How would this impact the rest of us?

Would Java Die?

Since we're all in the Java world, let's look at what might happen with Java. As we know, most of the major API development is performed on De Anza drive in Cupertino, at Sun. Now let's take Sun out of the picture. They're gone. Let's assume they lost all their top engineers, and their satellite offices didn't have the know-how to continue core development.

I don't think Java would die, but it would be a very different Java world from what we know now. Java would probably splinter into *x* different versions and go the way UNIX did back when it was released. Every major IT company in the world would assume responsibility and start supporting their new flavor.

Microsoft is a prime example of a company that would like (and is trying) to own Java; IBM is another. Each of these chaps has huge amounts to gain by controlling the Java platform. Microsoft fears Java will threaten its dominance in the PC marketplace, so it's try-

ing everything in its arsenal to taint Java and make it difficult for its programs to run. Anyone who's coded an applet for IE4 knows only too well that the same applet for Netscape will not run half as sweetly as in IE4.

Microsoft has dismissed the idea of a Java station as impractical, and is very keen to promote its WinTerm as an alternative. Chances are this will succeed. Why? Because, let's be honest, we have very little choice. Big corporations will stick with solutions already in place, and NT is gaining market share rapidly. Does that mean we as Java developers are going to have to face the Microsoft classes at some point in our careers? Hopefully not.

Now, assuming that in our game here Netscape is also taken out of the game means the browser war is effectively over. IE will become the de facto, nay, the only browser available. So we can conclude that taking Silicon Valley and removing it from the face of the earth would make a significant difference.

It's been fun for me to research this article. I've had conversations with people in San Francisco, New York, London and Sydney. Some were CEOs of Java companies, some were developers; some CEOs had nothing to do with Java but were major users of large installations of Java.

I'm not going to quote anyone here as that wouldn't be fair, but let me just say that it was easy to spot which companies didn't want to bite the hand that fed them. The ones proclaiming it wouldn't make a bit of difference were found to like Microsoft that little bit too much. The ones who said it would make a difference were the ones that had no alliance or allegiance to Microsoft at all. Not exactly hard science, you understand, but like I said, the answers I gleaned did make me chortle at times.

But It's a Game. It's Unrealistic. Or Is It?

Who remembers that brilliant movie *It's a Wonderful Life* starring James Stewart? Do you remember the storyline? Basically it's about a man, George Bailey (James Stewart), who's got to the end of his tether and wants to commit suicide. This is because all his life he's been trying to run a business under the constant strain of the mean Mr. Potter, the richest man in the county. Mr. Potter is determined to put George out of business and tries a number of tricks. Finally George gives up and can't see a way forward.

Do you remember what happens? Clarence, his guardian angel, comes down and shows him a world without George. The town is no longer called Bedford Falls but Pottersville. Everything is owned and controlled by Mr. Potter and the place is in a right old mess. So, in good old movie tradition, George sees the error of his ways, returns to his old life and discovers that the whole town has rallied around to help him out. It's a sweet story, but one that I couldn't help but see has some parallels for our own.

In our world we have Mr. Gates, who would be Mr. Potter, with Scott McNealy taking on the James Stewart role. With that in mind pick up a copy of the film and watch it again. The movie takes on a completely new meaning. Especially the scene where Mr. Potter tries to buy George off. Considering the current events in the media with Sun and Microsoft, it hits a recognizable note.

This dream has turned out to be a bit of a nightmare. A silly off-the-cuff remark sparked a whole lot of dialog that got everyone I was in contact with thinking about the bigger picture. Bottom line: if Microsoft got control of Java, it would lose one of its basic fundamentals – the ability to "write once, run anywhere." We as Java developers and keepers of the faith simply can't let this happen.

About the Author

Alan Williamson is the CEO of N-ARY Limited, a UK-based Java software house. He can be reached at alan@n-ary.com (www.n-ary.com).

alan@n-ary.com

The Most Important Features for Java Developers

by **Michel Gerin**

Delivering on the Promise of Java

Java would not have achieved the momentum it has today without delivering one key benefit: platform independence. Today, more corporations are facing the challenge of integrating disparate environments. This challenge is growing even more with the increasing number of mergers and acquisitions. NationsBank, featured in this issue of ***JDJ***, is a prime example of this integration challenge. Acquiring new banks on a regular basis, including its recent merger with Bank of America, NationsBank turned to Java to bridge its different systems and allow them to work together. When it comes to customers and assets, organizations need reliable, accurate and timely systems in place, and developers need the right tools to deliver those systems.

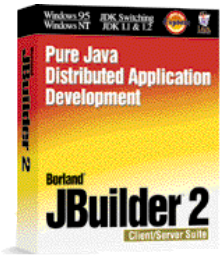
It's All About Pure Java

We created JBuilder, our award-winning rapid-application development tool for Java, to include features that developers have asked for most, including an intuitive user interface, a fast compiler, a complete graphical debugger, JavaBeans with source code, a reliable database architecture, easy creation of JavaBeans, tight CORBA integration and full support for Java standards. After talking to our customers, however, we discovered the feature they found to be most important was 100% Pure Java code creation for true platform-independent applications. This feature alone was key for customers like Oracle, NationsBank (whose application is highlighted in this issue), Daiwa Securities and others. These companies chose JBuilder after closely evaluating several other Java development tools. These other tools add proprietary code or markers to the existing Java code to synchronize the visual designers and the source code. If changed, this non-Java code will break the synchronization between the two and will make a developer's files unreadable to visual designers, thus causing developers to lose all their work. We created JBuilder to ensure that Java programmers have the most productive, reliable tool to deliver platform-independent, 100% Pure Java applications – no markers or proprietary code added!

> ***"...most important was 100% Pure Java code creation for true platform-independent applications."***

Infrastructure Is Key

Java developers also told us they needed a strong supporting infrastructure to ensure their success. Java wouldn't be successful today if it weren't for its huge infrastructure that includes major third-party partnerships, books, magazines, training, consulting, support, third-party tools, newsgroups, literature and Web sites. As with Java, JBuilder wouldn't be the market leader today if it weren't for its impressive supporting infrastructure. Before we shipped the first version of JBuilder, over 6,000 developers were already developing Java applications with the preview versions of JBuilder 1. Among them were the leading JavaBeans vendors such as KL Group; major software companies such as Oracle and IBM; corporate and independent developers such as Daiwa Securities and MicroAge; and leading universities like MIT, Purdue and UCLA. JBuilder's growing infrastructure is already the most impressive of all Java development tools and includes hundreds of third-party tools, dozens of books, an active Java developers' community that shares tips and techniques on over 20 dedicated newsgroups, a growing number of certified trainers and consultants, state-of-the-art technical support, multimedia training, an educational version (JBuilder University Edition)...and the list goes on.



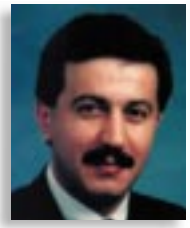
Evaluation Made Easy

Surprisingly enough, the third most requested feature for JBuilder was for an evaluation kit. A product may be the best development tool available, but if developers can't get their hands on it to evaluate it, chances are they won't even consider using it. Today, Inprise and ***Java Developer's Journal*** make it easy for Java developers to evaluate JBuilder. With this issue of ***JDJ*** you'll have the complete JBuilder 2 evaluation kit CD. The CD includes a 60-day trial edition of JBuilder 2 Client/Server Suite, which will let you build scalable enterprise applications; VisiBroker for Java, which will allow rapid building of multitier CORBA applications; *Referentia for JBuilder 2*, Volume I, an integrated multimedia training tutorial; an *Evaluator's Guide*; case studies, technical white papers, JBuilder tips and techniques; and more. The CD includes everything professional and corporate developers need to successfully evaluate a 100% Pure Java code-creation development tool.

About the Author

Michel Gerin holds an MBA in marketing and a BS in computer science. He is senior product manager for JBuilder at Inprise Corporation, and can be reached at mgerin@inprise.com.

mgerin@inprise.com



JBuilder
by Inprise

An IDE that features both power and ease of use

by **Ed Zebrowski**

Around the time Java made its mark on the world of programming there came to be another phenomenon, the IDE. Gone were the days of typing code, compiling, wondering what went wrong, typing more code, recompiling and so on. The IDE presented the application builder with an unprecedented tool, useful not only for application building but for learning as well.

I found I could learn more about Java code when I started using IDEs. I'd use the drop-and-drag editor to add something to an existing application, compile and run it, then look at the source code to see what I'd just done. It provided a neat little shortcut to learning more about objects, events and such. I've had the opportunity to work with most of the IDEs that have come our way in the past couple of years, and always tried to see what I could learn from them.

Recently I was given the opportunity to work with JBuilder from Inprise. Not only did I think it was a good application development tool, I found it exceptionally useful as a learning apparatus as well.

System Requirements

- Intel Pentium/90 MHz or higher
- Windows 95 or NT 4.0
- 48 MB of RAM (64 MB or higher recommended)
- 100 MB of hard disk space (minimum install)
- CD ROM drive
- SVGA or higher resolution monitor (800x600)
- Mouse or other pointing device

I installed on a Cyrix 150 with Win95 and 64 MB of RAM. The installation CD has an "Auto Run" file that brings up a menu as soon as the CD is placed in the drawer. The standard Installation Wizard made installing JBuilder as easy as pie!

Finding Your Way Around

I found JBuilder remarkably user friendly. Upon opening, I was greeted with a friendly "Welcome Screen" that, if necessary, can guide the user through a series of "Welcome Projects," as seen in Figure 1. These made getting familiar with JBuilder quick and easy. The projects consist of a couple of quick applications built as tutorials. After using them, even an inexperienced Java programmer will have the confidence to use JBuilder. Any last-minute pointers or changes from the previous version are presented in the Welcome Screen as well.

The Welcome Screen is displayed in what is known as an "App Browser," a window that allows you to "browse" through all the files and projects created with JBuilder. It allows you to browse, edit, design and debug these files. The App Browser contains three panes:

JBuilder
Inprise
100 Enterprise Way
Scotts Valley, CA 95066-3249
Phone: 408 431-1000
Web: www.inprise.com/jbuilder
Price: $2,495 (client/server suite version)

- ***The Navigation Pane:*** Located on the upper left of the screen, it displays a list of files that may include .java, .html, text or image files. If a project is made current, the Navigation Pane will also display the .jpr files (the default extension for JBuilder projects).
- ***The Content Pane:*** Located on the right side of the screen, it displays the detailed contents of the file selected in the Navigation Pane. This pane has various types of viewers at its disposal, the use of which depends on the file selected. For example, selecting a text file in

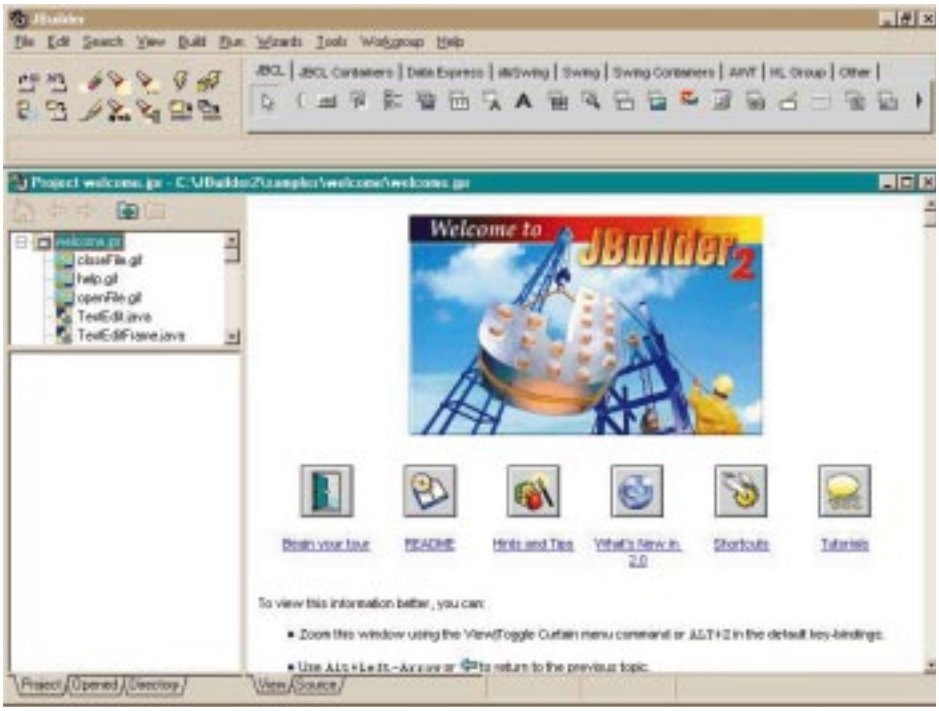


Figure 1: The user is greeted with a "Welcome Projects" screen, a guide through JBuilder.

the Navigation Pane will prompt the Content Pane to display that file in a text editor, similar to the Windows Notepad. If an image is selected, say a jpg, an image viewer is opened and displays the file. If the image needs to be changed before use in your application, you'll have to do that in another editor, as the JBuilder image viewer doesn't have editing capabilities. The Content Pane gets interesting when a .java file is selected from the Navigation Pane. In this case four tabs are displayed on the bottom of the pane: the Source Tab selects the JBuilder Editor, a syntax-aware programming editor that features several key mappings; the Design Tab calls JBuilder's UI Designer (see Figure 2), which shows how user interface of the class will look at runtime (here it's possible to visually construct and alter your application); the Bean Tab exposes the BeansExpress Property, Event, BeanInfo and Property Editor designers used to add properties and events to your bean, choose what properties are exposed or create custom property editors; the Doc Tab displays the corresponding reference documentation for that .java file if available (given in HTML format).

- ***The Structure Pane:*** Located in the lower-left pane of the App Browser, it shows structural information about the file selected in the Navigation Pane. If a .java file is selected in the Navigation Pane, for example, the Structure Pane will show imported packages, classes and interfaces on the file, ancestor classes, variables and methods (given in the form of a hierarchical tree). When one of the structural elements is clicked, the Content Pane will move to and highlight that element in the source code. This comes in handy as a way of finding elements in the .java file.

Above the App Browser is the Main Window, which contains all the necessary tools for application development. Among them are:

- ***The Toolbar:*** Provides shortcuts for some of the menu commands such as Open, Close, Save File, Undo, Redo, Search Replace, Run and Debug.
- ***The Component Palette:*** Displays components used in the design. Components can be added to the existing pages or new ones can be created for them. Some of these are JBCL, JBCL Containers, dbSwing, Swing Containers, AWT and the KLGroup.
- ***The Status Bar:*** Appearing at the bottom of the Main Window, it displays file save and compilation progress messages.

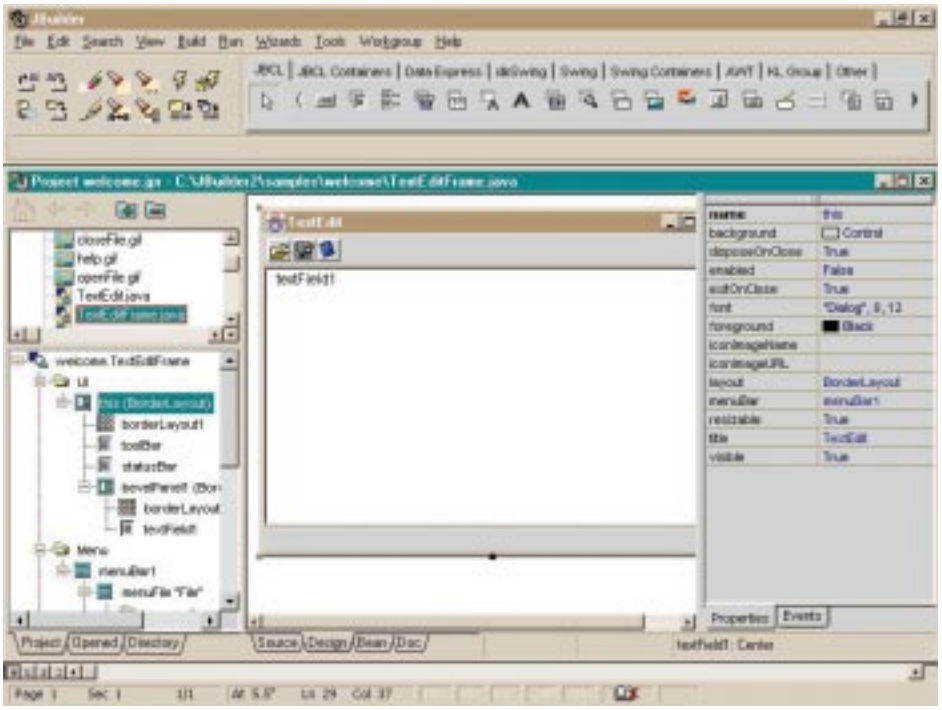


Figure 2: The UI designer makes it easy to add interactive components to your application or applet.

Using JBuilder

After taking a few minutes to get comfortable moving around in JBuilder, I found it just as easy to develop applications in it. Let's develop a quick, basic application just to get a feel for how easy it can be to use.

First you open a new project by clicking File-New and then double-clicking the Application icon. There's an icon for applets too, if that's your choice. This will open a simple dialog box that contains five fields. In the top field type the entire path to the location of the project files (for example, C:\JBuilder\myprojects\project1\project1.jpr). The other four fields are for title, author, company and a brief description, if you so desire. After these fields are filled in click Finish, which will generate the .java files that go into the project "skeleton." If the applet Wizard is used, a corresponding HTML file will be generated as well. Clicking Finish on this dialog box will then open the Step One Wizard dialog box, which requires you only to enter the name of your class in the appropriate field. There are two checkbox options: "Use only core JDK and Swing classes" and "Generate header comments." Clicking Next opens the Step Two dialog box. If you're building an application rather than an applet, here's where the window properties are set. The checkbox options here include "Generate menu bar," "Generate tool bar," "Generate status bar," "Generate about box" and "Center frame on screen." Upon clicking the Finish button, new .java classes are added to the project.

Now it's time to use the UI (user interface) Designer. Clicking the newly created .java file in the Navigation Pane will bring up the source code in the Content Pane. Clicking on the Design Tab will bring up the UI Designer. Here it's possible to assign menu options, generate text fields and add buttons or dialog boxes. Events and properties are easily assigned by clicking the appropriate tab on the Component Inspector on the right-hand side of the Content Pane.

More Than Just a Beginner's Toy

Up to now, I've gone on about how easy it is to use JBuilder. Don't let this mislead you. Although JBuilder can be very simple to use, it includes a powerful arsenal of tools for the advanced builder as well. Some of the more sophisticated tools allow you to:

- Create distributed applications with Java RMI.
- Define CORBA interfaces with Java.
- Create data-aware components.

It's getting to the point where choosing an IDE can be a mind-boggling task. It's really impossible to sit down and try them all. JBuilder is an excellent choice for both the beginner and pro. Use it for a while and you'll agree; there's no need to look any further.

About the Author

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

zebra@rock-n-roll.com

CASE STUDY

NationsBank's 100% Pure Java Solution

by John Melka

JBUILDER FOCUS

A bank's journey from legacy systems to Java

NationsBank develops a Java solution, using JBuilder from Inprise and a variety of other Java tools

About the Author

John Melka has over 25 years' experience in design and development engineering in multiple disciplines. He currently works for NationsBank Services, Inc. He can be reached at johnm@crt.com.

johnm@crt.com

I wish I had a nickel for each time I've had to explain to a new vendor or professional acquaintance that, just because I have the word "Bank" in my company's name – NationsBank – I don't spend my time dealing with huge, monolithic, batch-processing systems. (Sure, we have these systems. And yes, they're still in use.) However, there are large groups within our bank whose purpose is to explore the leading edge of technology. Our work centers on incorporating this technology, both tactically and strategically, in production systems. As most of these systems are for internal use, few people see them unless they work with us.

Evolution of the Bank's Information Technology Departmental Structure

While the details of the systems we develop are confidential (they're part of our competitive business advantage), I'd like to share with you our experiences concerning architecture, tooling and how our development environment has evolved. To do this properly, I first have to explain who we are within the bank structure, how we started and how we've evolved. Next I'll examine how our architecture has evolved into its current form. Finally, I'll discuss the tooling we use to develop this architecture and how I envision it must evolve from today. Please remember that this is a work in progress. What I'm describing here is the culmination of years of work and is presented with a deliberately blind eye to any missteps along the path to here.

The Global Finance Software Engineering Group supports the Global Finance (GF) section of NationsBank. NationsBank is divided into two sections. *General Bank* is the section of the bank you and I use for our checking and saving accounts or to get and pay a car loan. General Bank is the largest part of NationsBank.

Global Finance is the second section. As large corporations, say $250,000,000 in sales, have different needs from their bankers, Global Finance has a distinctly different set of services. These needs dictate different performance requirements – and usually much more exposure to the external customer. Consequently, Global Finance has chosen to use technology to leverage an advantage in this arena.

My group, IS Tools, is part of the Global Finance Software Engineering Group. We evolved from the 1993 acquisition of Chicago Research and Trading (CRT) by NationsBank. Between 1993 and 1996 NationsBank explored this new acquisition. In addition to the "derivatives" (specialized combinations of options, futures and indexes that can be purchased and traded) knowledge the bank had obtained, CRT had developed and refined a method of high-tech rapid development to support derivative trading. NationsBank, realizing this was a potential competitive advantage, asked the CRT group to help implement their rapid application development (RAD) process within Global Finance as a whole.

To facilitate this, the head of IT from CRT asked to join NationsBank Services, Inc., to support Global Finance as the head of software engineering. As no group within the Services company dealt with development tools at that time, we, the IS Tools Group, also moved to support Global Finance. The manager of IS Tools then became the head of a team named Technical Architecture and a manager of IS Tools in Chicago was selected from the Tools group.

We then created a tool group, with a local manager of IS tools, for each of our other main development areas, Charlotte, North Carolina, and Dallas. This provides a more locally accessible point of contact for the developers in each region. The three managers report to the Technical Architecture Team Lead, who in turn reports to the head of software engineering. Each local manager supplies support, specialized expertise and, indirectly, training for each of the supported tools, and vendor contact and contract support coordination.

Evolution of the IT Architecture

This arrangement shows our appreciation of the significance of architecture, and the infrastructure to deploy it, within the Global Finance section of the bank. It also underscores the importance of the use of technology in lowering our costs and increasing our efficiency to our customers, the developers who support the decision makers within Global Finance.

Now that you know our position within the bank, and how we evolved here, let's dis-

cuss the evolution of our architecture, and what tools and technologies we've used to accomplish this evolution.

Over the years we've watched our architecture change from the monolithic to traditional client/server. When we lost power with traditional client/server, we moved to light client/server. Light client/server has progressed to the three-tier architecture (a heavily misused term). And we're finally moving to the fully distributed n-tier.

The monolithic is often forgotten in our evolution discussions. It formed the basis for all our batch programming and exists (mostly as "legacy" code) for many purposes even today. It still fills many purposes for which it is well suited (most of us are paid by such a program). As this technology proved itself to be brittle and hard to maintain, it served to launch us into the next phase – traditional client/server.

In traditional client/server each delivered function has its own client and its own server. If the organization is really astute, there is a set of standards that defines the user's interaction with the client, i.e., menu structure, what mouse clicks do, drag and drop, and so on. Hopefully this group also followed the standard that the OS developers used. (This assumes that the OS that clients were developed for HAD a standard at the time the clients were developed.) The second best is that the interfaces are REALLY different. If they're actually only close, you usually want to retrofit your clients with the OS vendor's system, no matter the expense.

An Outgrowth from Traditional Client/Server

We began to outgrow the traditional client/server when the phenomenon I call "desktop implosion" occurred. When you begin to get more and more functionality on the desktop, the number and interaction of clients begins to balloon. As the ballooning proceeds, the density of programs eventually reaches critical mass and the desktop becomes a black hole, sucking the entire content of the office (and the hapless user) into it. (Actually, the number of programs and interactions destabilizes the desk and becomes a nightmare to update, but the black hole metaphor is much more graphic.)

Client/server does, however, have some advantages. We can tune all of our server applications for maximum performance. We can set up special auditing, tracking and accounting information within each of the servers. Against that we have a number of disadvantages.

Disadvantages of Client/Server

The first disadvantage is the deployment problem. When the crews paint the Golden Gate Bridge, they start on one end and paint toward the other. When they reach the other end, the beginning is peeling so they start all over again. This continues ad infinitum. If you imagine that they change color each time they run out of paint, we have an analog to our deployment problem. At no time will they have a stable color for their bridge, just as we won't have a stable configuration on our workstation or servers when we reach the critical number of applications.

Second, we have little reuse of software except within libraries. These are generally developed for one application and then "upgraded" for future inclusion. While this reuse is minimal, the practice is on the right path. It's usually less effective as the number of platforms increases.

A Need for an N-Tier Approach

Our first attempt to alleviate the traditional client/server problems is to convert to the light client/server. We move the client code to the server and combine it with the server code. We use a "general client" with a "glue" language to keep the user interface on the workstation. Our general client is a browser. The glue language is HTML. This setup is what some people call "three-tier" with the database forming the third tier. As we'll see, this usage has confused three-tier with the simpler light client/server.

We have now alleviated the deployment problem. We deploy to a server (or server farm) that is significantly simpler than a large number of desktops. This improves our scalability problem. We can still collect our audit and tracking data, but now we rely on an authentication method to really ID the user (either the CGI type authentication or some "single sign-on" technique). We do need an investment in the infrastructure because we establish an intranet (using either Web technology or our own) to create the link. (We did some of this in traditional client/server, but it increases with the utilization increase of light client/server.)

Multiple Platforms Come into the Picture

Our downside here is that we may have to design for multiple platforms and browsers. When we reached this phase we had multiple versions of multiple browsers on two different platforms. Since placement is difficult without the same resolution on each desktop, to say nothing about different browsers, our testing time increased dramatically. We alleviated this problem by specifying the kind and version of browser on each desk.

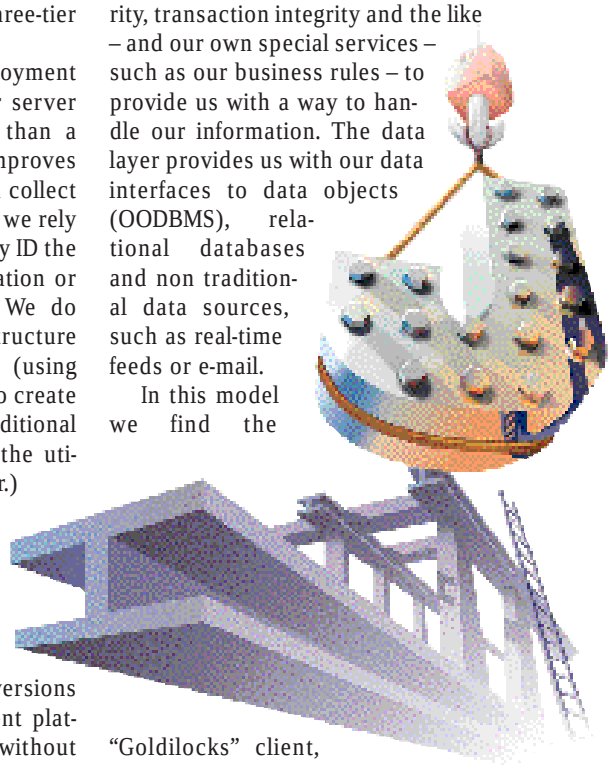
We still have minimal reuse of this model. In addition, as HTML is stateless and not really designed for this purpose, we've used JavaScript or VBScript for any user feedback. Not using these techniques results in the "send it and find out how it breaks" pages we've all come to hate. As this method requires an HTML generator (or generation logic), we have to learn these tools. This results in a flatter productivity versus time curve for the developers entering into this design model.

While this technique is better than the traditional, we still eventually run into a complexity issue. The new model addresses the deployment problem but does it at the expense of the productivity curve. To alleviate this problem we have to address the reuse problem and find a way to blend a number of our functions. The blend allows us to add functionality by adding only the changes necessary to create that new functionality.

Implementation of a Three-Tier Architecture

This blend becomes our three-tier architecture. The three tiers we refer to here are the presentation, server of services and the data layers. The presentation layer is our interface with the user. It gets what information the other layers need and returns the results they send to us. The server of services provides us standard services – security, transaction integrity and the like – and our own special services – such as our business rules – to provide us with a way to handle our information. The data layer provides us with our data interfaces to data objects (OODBMS), relational databases and non traditional data sources, such as real-time feeds or e-mail.

In this model we find the "Goldilocks" client, usually a browser augmented with one or more plug-ins, that's not too thin, not too thick, but just right. On the server side we incorporate a method of implementing the server of services so it can be spread across a number of servers. This gives us the scale we got in the light client/server without having to replicate the code across the multiple servers. We now have a more encapsulated object model, so



we can reuse at the object level.

Our downside is that we have to provide more infrastructure to implement the design. In addition, we've altered our deployment model. While we don't deploy to multiple servers, we have to make sure we deploy objects compatible with existing objects and applications. This makes interface design and maintenance very important. Last, our design is still predominantly client/server.

A Move from Three Tier to N-Tier

Our latest move is to expand the three-tier model to the n-tier model. In this model we expand the ability to tier back to the client. Both client and server use objects to implement functionality. In our browser example CORBA-enabled applets can serve this function.

On the server side the middle tier stratifies into interface, business and data strategies. Objects, both general and specific, implement these strategies. At this point component architecture comes into play because the component provides the model for reuse.

This is where we're at with our architecture. We have applications that embody all of the steps along the way (with the exception of monolithic, I think). With our discussion of who we are, you should have a good idea of why our architecture developed the way it did.

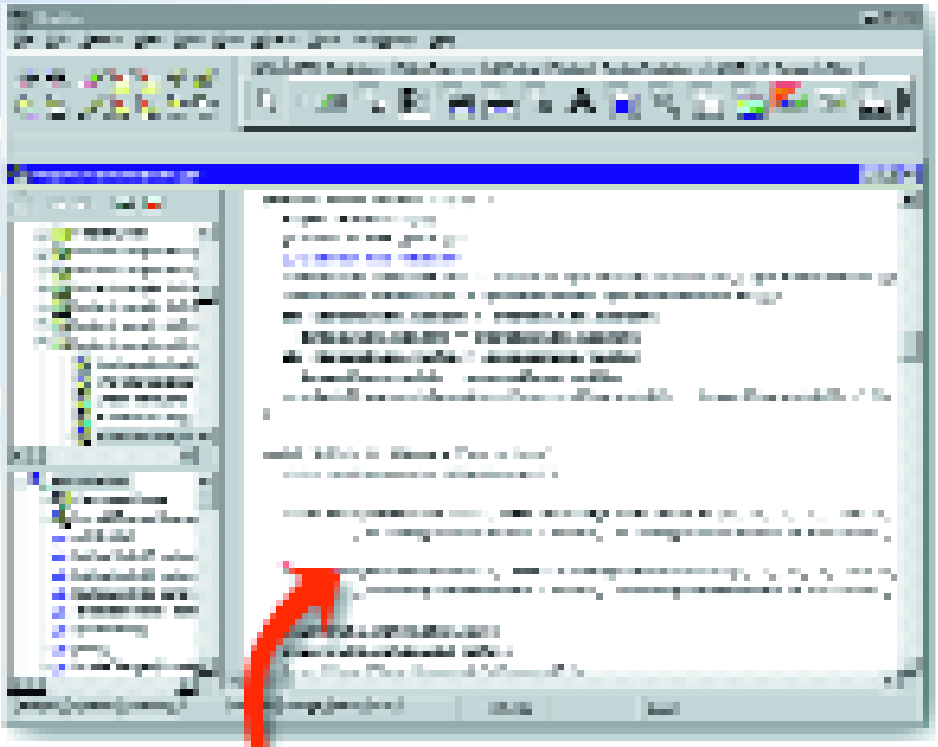
A New Set of Requirements

Performance: A whole set of requirements falls out of our architectural experiences. First, you'll need levels of performance in both development and production. The performance is necessary because, in the rapidly moving environment we all work in, asset RAD is necessary to ensure time to market (even if the market is internal). If you don't believe this, see if you can remember when development cycles were 18 months.

Internet: Departmental projects have needs different from those of general Internet applications. Internet applications require a whole security infrastructure to ensure enterprise integrity. Internet applications have needs different from those of intranet applications. Our internal customers are usually more demanding.

Extranet: Extranet applications have a set of needs different from those of Internet or intranet applications. In the extranet arena you have to protect yourself from external intrusion while ensuring that your customers are the only people getting your content.

You need to examine BOTH applets and applications. Most people look only at applets and forget that the browser-launched application can provide better functionality than an amalgam of applets and HTML. You have to look at the design from all aspects.



100% Pure Java Code
No proprietary code
or markers

Platform Issues: Platforms can be both a liability and an asset. Some platforms lose scalability before others. You need to design for flexibility so that redeployment doesn't involve a part of the software.

Software Configuration Management (SCM) is essential for viability. You must be able to reproduce your work despite turnover, revision or disaster. This requires both discipline and infrastructure investment, but it's a vital link and shouldn't be scrimped on.

Frameworks: Frameworks are essential. Without frameworks you can't achieve the uniformity necessary to reuse objects and components in the environment. Frameworks provide functionality that is necessary but not core to your business. You can purchase a number of these frameworks and save a lot of time and effort. Make sure you get support with them.

Distribution: Distribution is essential. You need a means of distributing your objects and components. If you believe this is an easy problem, remember how many times your Windows 95 system has crashed because some installation program has decided to overlay a new DLL with an old one.

Testing: Testing is different in this environment. As your network is now part of your environment, your testing must include it. This means that you not only need to test on every platform your application could possibly be used on, you must also load-test across the network. Measure all the loads you expect the project to be used at. Then continue the test to overload so you know when to upscale the implementation.

A browser must be part of the configuration. If you can, specify one type of browser and restrict the number of plug-ins to a reasonable number. This lowers the desktop update requirements.

We want to open standards to achieve our goals. We define open standards as ones in which the specifications are published by the promulgator and at least two vendors supply product. While you usually cannot adhere 100% to open standards, you need to control when you deviate and what you lose by that deviation.

Java as the Solution

We chose Java to provide our cross-platform capabilities. We have experience with C++ in multiple environments and know how expensive it is to maintain. If Java lowers this cost, it is a significant saving. In addition, we have the option of creating applets with Java.

CORBA supplies the middleware. CORBA is platform- and language-agnostic. This allows us to decide how and what is used to implement our applications. In addition, the

interface description language (IDL) provides an ideal, compileable medium for interface specification. The CORBA component, transaction and security models provide the essential services necessary to build our applications.

C++ provides the high-performance parts. Most of us have a large body of custom C++ code that we have no desire or, sometimes, ability to replace. JAVA is not a high-performance mathematical language – yet. Because of our choice of CORBA, we can be agnostic about which language components are constructed from.

JBuilder as the Java IDE

For our Java IDE, we chose JBuilder from Inprise. We are already a Delphi house and knew how to use the environment. Since we had a strong NeXTStep background, we wanted a strong component development environment. We had already seen this in the Delphi environment. With JBuilder's implementation of JavaBeans as their component model, we felt confident in our choice. Our experience with switching from code to graphics and back, seamlessly and without encumbering comment tags, has enhanced the speed and reliability of code development. 100% Pure Java generation means that I have to generate the code once and simply test it on each of the possible deployment platforms. This is a significant saving of labor versus either C++ or designers that use custom heavyweight components.

VisiBroker for CORBA Implementation

VisiBroker provides the CORBA implementation. It has excellent Java and C++ implementations. It is well integrated with JBuilder and thus is easier for developers to use (they don't have to leave their environment to use the tool).

PVCS, NetDynamics and Mercury Interactive as Additional Tools

PVCS provides our SCM. It is integrated with JBuilder and also works with our server-side Java framework, NetDynamics. It was already a standard within the bank. This made deployment and support infrastructure consistent with our other development environments. It also supports both ASCII and non-ASCII code parts.

NetDynamics provides the server-side Web development framework. As HTML is the glue layer in our presentation layer, we need a robust server-side generation engine. We extensively examine the framework market on a regular basis. In our environment NetDynamics consistently exceeds the others as a production system. The incorporation of VisiBroker CORBA as their ORB makes for a wide CORBA pipe from our application presentation layer to the back end.

Rounding out our development environment are our Mercury Interactive test tools. These tools were chosen because the test scripts written at any point in the test procedure can be utilized at other steps. Thus the unit tests can be used as part of the integration test, and so on. In addition, their load-testing tool can be used to properly test NetDynamics.

What we see as needed future directions encompass test tools, remote debugging, a more general component model, application-specific and general servers, and more commercially available components. The test tools should give better insight into the applications function at the JVM level as well as provide code coverage and performance analysis. Remote debugging should be capable of debugging code anywhere on the network. Servers should provide support for the Enterprise JavaBeans and CORBA component models. Components should evolve from the basic "list, queue and dequeue" models.

I hope our experiences at NationsBank provide some insight into how and why our architecture has evolved and what tools we've chosen to implement this architecture.

Referentia for JBuilder

Integrated Multimedia Training for JBuilder Professionals

by **Larry Lieberman**

One of the most frustrating and expensive aspects of transitioning to any new software product or version upgrade is the impact to productivity while tackling the learning curve. The downtime associated with learning new tools and techniques can be a significant barrier to technology adoption, and many firms will lag with the status quo to avoid the potentially cumbersome implementation of new products. The product team at Inprise took this factor into account when planning the launch of JBuilder 2, and adopted an integrated learning system from Hawaii-based Referentia Systems Incoporated, to provide a rapid ramp-up for the new release.

Taking full advantage of JBuilder 2's open API and architecture, Referentia adapted the Referentia Learning System to provide an integrated, extensible multimedia training resource that JBuilder users can access 24 hours a day. At its core the system is a software framework designed to integrate with software applications and deliver multimedia instructional material. This new learning platform offers rich concept and lesson animations, a keyword topic search for performance support on the job and an advanced "Try It" feature, letting users load-sample files directly into JBuilder and practice the lessons along with step-by-step narrated instruction. The integration with JBuilder 2 includes links to JBuilder Help documentation, accessibility through JBuilder's Help menu and hooks to call JBuilder to the foreground in the "Try It" mode. With content focused on project examples and on-the-job productivity tips, the system is designed to bridge the gap between online reference and full-fledged training.

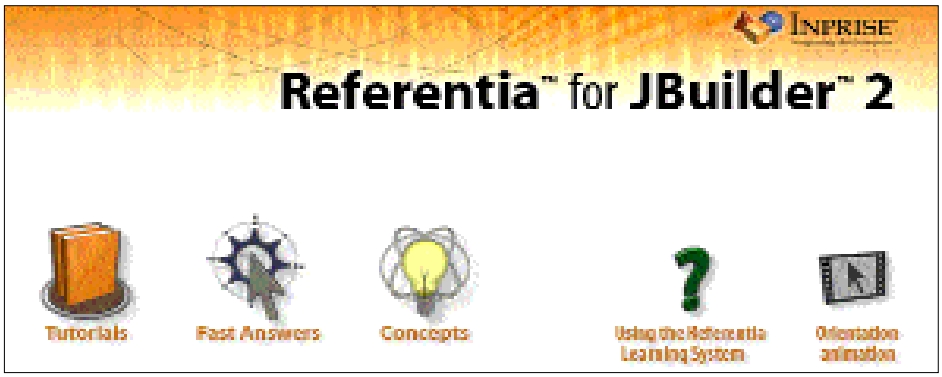


Figure 1: Tutorials, Fast Answers and Concepts are the three main sections of the Referentia System.

The idea isn't new -- in fact, for many years visionaries in the fields of computer-based training and electronic performance support system design have suggested built-in training for software products as the obvious ideal in a multimedia-capable world. Only in the last couple of years, however, has the installed base of multimedia-capable workstations grown to levels that can provide a firm market for rich audiovisual training aides.

Referentia Systems has been at the forefront of the integrated multimedia training field, supplying Autodesk with an in-the-box integrated training system for AutoCAD Release 14 in May 1997. The product, first of its kind, won awards in both the CAD industry and in tech pubs/documentation circles.

The newest version of the Referentia Learning System adds the ability for multiple volumes of training content to be introduced into a single outline tree. Referentia for JBuilder 2 leverages this capability by categorizing the available content into separate CD-ROM volumes, thus giving users the opportunity to choose from a library of content titles geared to discipline-specific needs.

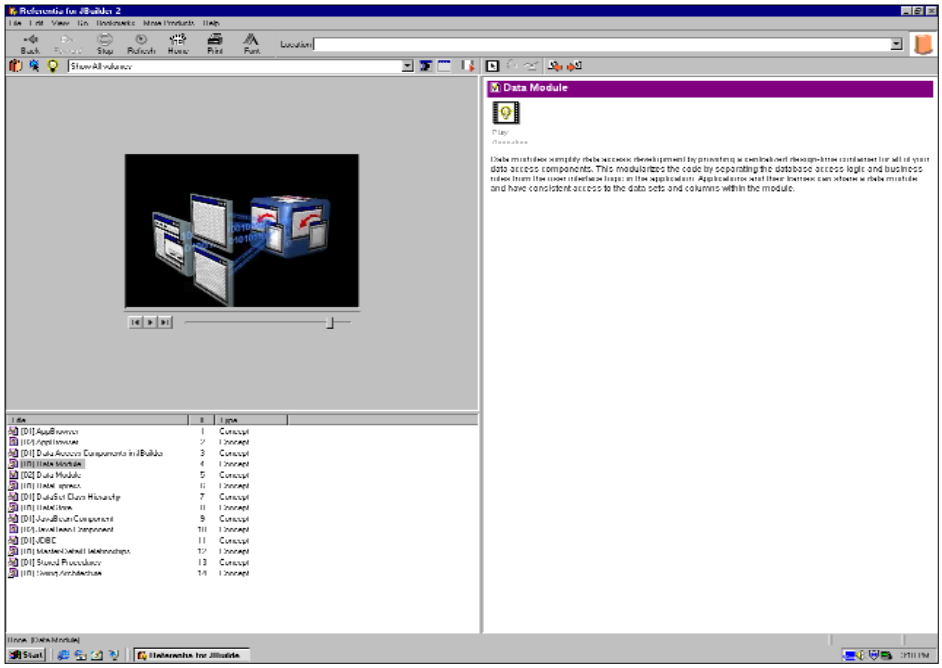


Figure 2: Watch and listen to step-by-step lesson animations, then try it in JBuilder.

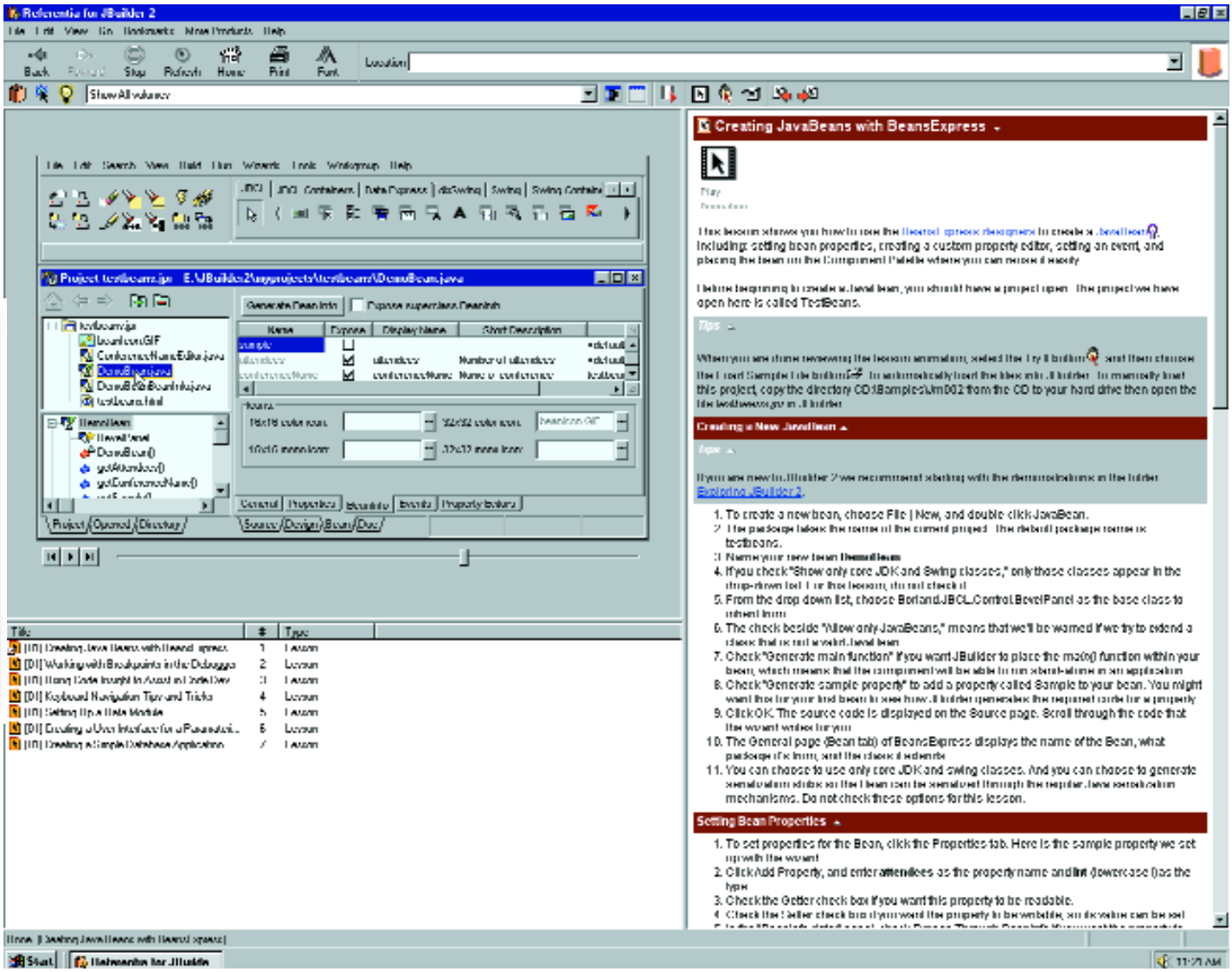


Figure 3: The Try It feature leaves an instructional window on top of JBuilder.

Volume I: Getting Productive

Volume I provided with JBuilder 2 contains a starter set of content including tours of the JBuilder 2 IDE and quick-start lessons for building JavaBeans, setting breakpoints in the debugger, creating a UI for a simple database application and other helpful information.

Volume II: Building Applications

This is a full-sized Java database application development course containing roughly a week's amount of instructor-led training content on a single CD-ROM. More than 45 in-depth lessons and concept animations are designed to help users increase productivity in developing real-world Java database applications. In this series of interactive tutorials, users gain a strong understanding of the JBuilder data model, master a solid foundation of essential and advanced database techniques and learn how to complete their database application development with advanced user interfaces and various deployment strategies. Specific topics include terminology, relational databases, JDBC, DataExpress, DataStores, master detail forms, queryResolvers, queryProviders, layout managers, JFC/SwingSet, Web deployment and other key topics.

Volume III: CORBA Essentials

Volume III will cover the basics of distributed computing in JBuilder 2, including deployment server and enterprise beans. It's scheduled for release in early 1999.

Once installed, Referentia for JBuilder 2 is accessible from the desktop or through JBuilder 2's Help Menu. The system can be run as a stand-alone training tool on a multimedia-capable PC or integrated with JBuilder to provide performance support on the job. The system's browser-style interface offers three functional areas of navigation: Tutorials, Fast Answers and Concepts.

The Tutorials section contains lesson animations grouped into a sequential tree outline. Opening any of the lessons reveals an HTML text window displaying the lesson steps, with hot links to JBuilder 2's online documentation for more comprehensive reference on specific topics or links to Concept animations for general reference. Selecting "Play Animation" starts an AVI sequence that brings the lesson steps to life in a smooth screen recording with accompanying step-by-step narration.

In the Concepts section users can choose from a glossary of terminology illustrated by rich 3D visualizations. The Concept animations offer strong reinforcement of complex subject matter (e.g., data modules, JDBC, Swing architecture) to allow users to grasp the big picture before they attempt the details of a lesson. Fast Answers is where users can look up information quickly through a keyword index or by category. Searches pull up information from all available sources Including relevant Tutorial lessons, Concept animations and JBuilder Help files.

The integration with JBuilder is key to Referentia's functionality. Although the system can be run as a stand-alone training

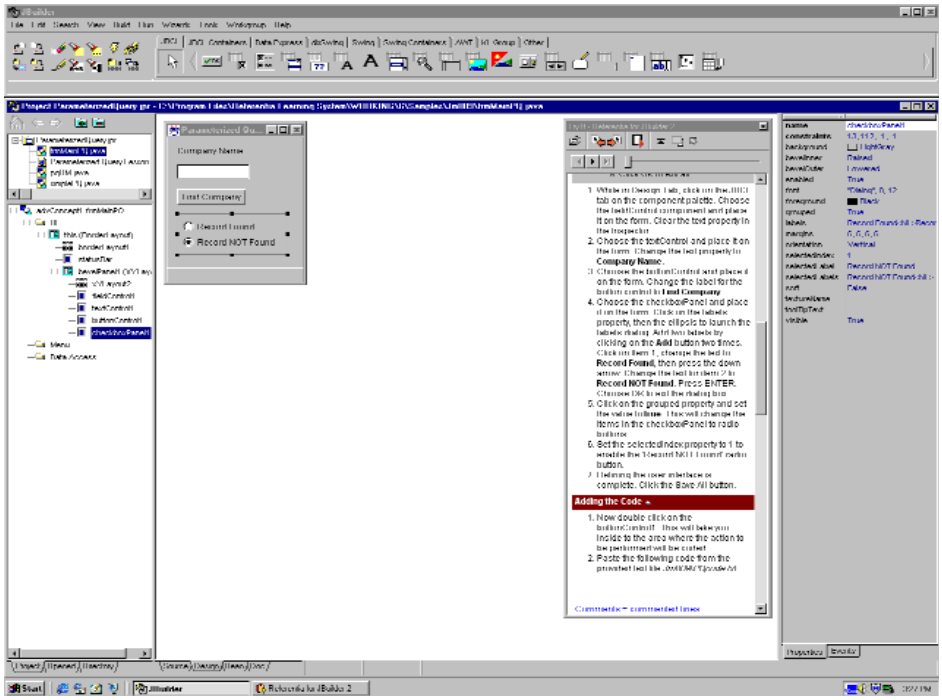


Figure 4: Concept animations bring complex terms and techniques to life.

aid, its optimal use is clear when running simultaneously with JBuilder 2. From any lesson, users can access a "Try It" feature that brings JBuilder to the foreground and leaves a floating instructional window on top with the lesson's step-by-step text. An "Open Sample File" button automatically loads project files into JBuilder for practice in real time. Once the files are loaded into JBuilder, users can follow along with the text or elect to play the audio track and listen to the narrated lesson steps while performing the operations live in JBuilder.

It was this live "Try It" feature that attracted technology scouts from the Ohio DOT to the project. Senior programmer Angelo Serra heard about the project through consultants working with Inprise. After testing the Referentia system in its beta release, Serra knew the product would mean less downtime for their team as they transitioned to Java and JBuilder.

"At the Ohio DOT we're including Referentia for JBuilder in our training program because our developers prefer the audiovisual approach of the multimedia CD," Serra said. "It will significantly increase our productivity."

Senior computer scientist Vic Askman at GTE Internetworking echoed these sentiments after incorporating JBuilder 2 and reviewing the accompanying Referentia Learning System. "The capability of the Try It feature is extremely useful, and the animations are well paced -- not too fast or slow," Askman said. "Content was relevant to Java as well as JBuilder 2, and it should have a positive impact on the bottom line since it will help ease the transition to JBuilder 2."

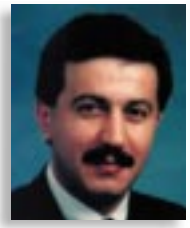
This new way of learning will help speed the development and deployment of Java applications for new and experienced users alike. Content is written to an intermediate level, assuming basic knowledge of programming in a RAD environment. Programmers migrating from other development languages will find that JBuilder's inclusion of an integrated multimedia training system allows for a rapid ramp-up on Java development within the JBuilder 2 IDE. Those with intermediate to advanced Java programming expertise will find the system useful for getting familiar fast with the new features of JBuilder 2. Future volumes of content will address specific JBuilder training needs from novice to advanced topics.

Referentia has supplied learning systems for AutoCAD Release 14, AutoCAD LT '97, Autodesk's Architectural Desktop and Seagate Software's Crystal Reports 6.0. Parties interested in evaluating the Referentia Learning System can request a free copy of the Referentia for JBuilder Volume I CD (shipping and handling fees will apply) and get more information about additional volumes online from Referentia Systems Incoporated, at www.referentia.com/jbuilder or by calling 1 800 569-6255.

About the Author

Larry Lieberman heads up corporate communications at Referentia Systems Incoporated His recent speaking engagements include technology presentations to the U.S. Navy and a conference address on the emergence, utility and impact of multimedia training materials. He can be reached at larryl@referentia.com.

larryl@referentia.com



PARTS for Java Professional Edition by ObjectShare

A great choice for a team-oriented development tool

by **Ed Zebrowski**

The company has just landed its most important account to date: a big client has signed a contract for the development of a Java program. It's an extensive application, requiring database connectivity and the use of ActiveX controls. You've been chosen to head up the project, which will require a team of developers. It's now time to choose an IDE that combines ease of use, powerful development tools and a team-oriented interface. With the increasing number of IDEs on the market, it could be difficult to choose one that fits this scenario.

I've just used PARTS for Java from ObjectShare. I found it to be smooth, powerful and full of necessary features.

System Requirements

- Intel 486 or higher processor
- CD ROM drive
- 260 MB free disk space (FAT 16)
- 90 MB free disk space (FAT 32)
- 24 MB RAM (32 MB recommended)
- SVGA graphics card
- Win95, NT 4.0 or higher
- TCP/IP installed and configured
- JDK 1.1-capable browser

Any earlier version of PARTS for Java must be completely removed before the current version can be installed. The CD-ROM has the AUTORUN feature and the Installation Wizard has a checklist of a few optional features. These can be installed later if you choose not to include them during the initial setup. The sections that must be installed are:

- Parts Pro Program and Documentation Files
- Swing Version 1.01
- JDK Version 1.1.5 (if not already installed)

I installed on a Cyrix 133 with 32 MB of RAM. Installation went smoothly and took only a couple of minutes. The application ran very well and didn't hang up the machine, as some IDEs occasionally do.

PARTS for Java opens with a Project Manager window, as seen in Figure 1, that acts as the "command center" for the IDE. It's the main window where all the elements of a project are organized. Along the top is a toolbar to launch all PARTS for Java tools.

I thought the Project Manager was nicely laid out, avoiding the confusing multiple-window fiascoes given by some IDEs. The toolbar enables the user to quickly and easily:

- Launch any applet or application.
- Compile whole projects or selected sections.
- Manage versions of projects and files.
- View the nested parts hierarchy in a JEDT file.
- Browse through any .jar or .zip file.
- Edit default path settings and/or descriptive comments for any project.
- Launch the other tools and wizards.
- Generate documentation by running JDK's javadoc program.
- Print information pertaining to the project.

PARTS for Java - Professional Edition

ObjectShare
16811 Hale Ave., Suite A
Irvine, CA 92606
Phone: 949 833-1122 Fax: 949 833-0209
Web: www.objectshare.com
E-mail: info@objectshare.com
Price: $1,495 download (includes OracleLite, OrbixWeb,Install Anywhere Now!) Standard $749, Lite $149

The window is divided into two panes. The left pane shows the hierarchical view of a project and subprojects. In the right pane is a report view of the project files currently selected in the left pane. When a project is opened or created, it's added to the left pane. New projects can be created as independent projects or subprojects of existing ones. Projects can be placed under version control. Different versions of a project are stored in a repository and can be retrieved or deleted at any time. A

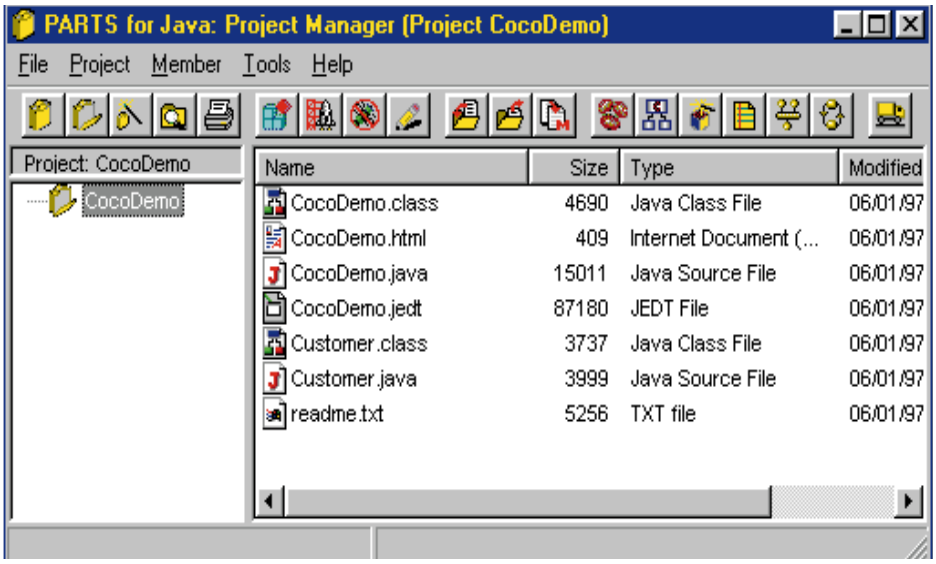


Figure 1: PARTS for Java opens with a Project Manager window, making it easy to organize and oversee any project.

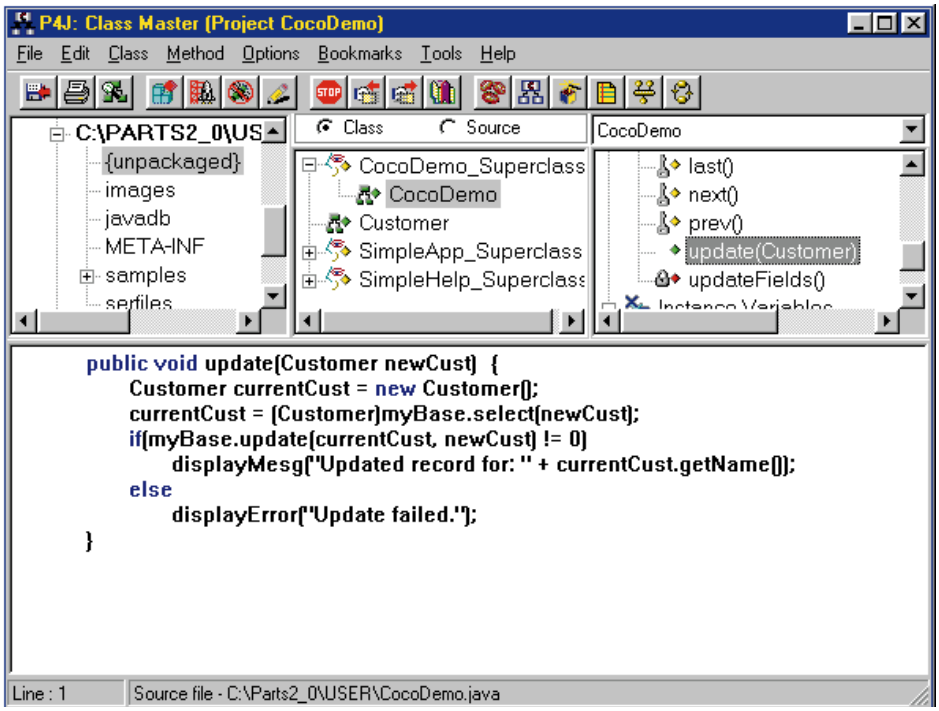


Figure 2: The Class Master allows the building of more specialized or complicated applications.

project may be "unversioned" as well.

It's also possible to change the configuration of PARTS for Java from the Project Manager window by selecting Tools-Settings Editor; a two-tab dialog box is displayed. On the System page it's possible to:

- Change the User Name settings.
- Change the Java Compiler, the Java Interpreter or the Applet Viewer.
- Alter the Java invoker port number. This helps avoid conflicts with other tools.
- Invoke the Show DOS Prompt During Launch option, which displays DOS windows containing status information about an application or tool.
- Check the Conserve Memory option; useful in systems with less than 32 MB of RAM, this option will increase the frequency of "garbage collection" operations.

When the Visual Designer page tab is clicked, the following options may be selected:

- Show Hints, Show Labels with Links, Show Links with Selection and Orthogonal Link Creation.
- Change the Grid settings so component parts may be easily aligned.
- Change Link Colors.
- Change Background Colors.
- Change Mouse Settings.

After installation and configuration, PARTS for Java is ready to create new applications. Selecting File-New Project will launch a dialog box. Filling in the appropriate information and clicking OK will add the new project to the Project Manager window.

The Visual Designer: A Complete and Powerful Tool That Has Drop-and-Drag Ease

Selection of a file in the Project Manager window enables you to initiate the Visual Designer. This is an object-oriented development tool that allows quick construction of applications by dropping and dragging prefabricated parts. These parts are displayed in a catalog window and include:

- A catalog of AWT parts
- A Swing catalog
- An ActiveX catalog
- A JDBC catalog

By using the Visual Designer I constructed a user interface for an application in only a few minutes.

The Class Master: When You Need More Complex Operations

The Visual Designer was excellent for rapid-fire development of a basic Java application. Some clients are going to have special needs, however, which may entail highly specialized operations that won't be found in the Visual Designer menu. When such an occasion arises, PARTS for Java has just the solution: the Class Master (see Figure 2). The Class Master opens as a four-pane window:

- The Packages pane in the top left is a tree view that shows the hierarchy of packages on the selected class path.
- The Classes pane in the top center is a tree view of the classes or an alphabetical list of .java source files.
- The Methods pane in the top right shows methods and variables when a compiled class file is selected in the Classes pane.
- At the bottom is the Source pane, which features a source code editor that allows the examination, editing and compilation of Java source files.

In addition to the Class Master there's a wide selection of other powerful tools. Among them are:

- A CORBA Wizard
- An RMI Wizard
- A Breakpoint tool
- A Debugging tool

As you can see, in addition to its ease of use, PARTS for Java is a powerful and professional development tool. If you're looking for the perfect tool for that big project, PARTS for Java is well worth the price.

About the Author

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

zebra@rock-n-roll.com

Java Virtual Machines

A discussion of three JVMs

by **Ajit Sagar**

The key selling feature of Java is its WORA (write once, run anywhere) promise. Let's pause and think about what's involved in making this promise a reality. "Write Once" is a concept that applies specifically to the Java language, the idea being that there is one and only one standard definition of the programming language that developers use for writing application code. In terms of syntax and semantics, this means the definition of the language is fixed, and any changes are routed through Sun Microsystems, the official owners of the Java Platform.

Providing a standard definition for Java's syntax and semantics is a feasible proposition. The "Run Anywhere" part of WORA is a much harder goal to achieve. "Anywhere" refers to the combination of hardware platforms and operating systems on which software written in the language can run. This mandates platform neutrality. All code written in a software programming language is compiled down to machine (assembler) code that is specific to a particular hardware platform; that is, the resultant machine instructions run on a particular CPU. Thus, to have the code run anywhere, the same source should compile to different machine instructions.

The Compilation Process

One way of translating programming language source code into native platform code is to provide the compiler with flags or directives that cause it to compile to a specific instruction set. However, this necessitates that the programmer or installer of the code provide the directives, depending on the hardware environment. Customizing source code from a high-level language for multiple platforms is not an easy task. Also, the precompiler directives have to be embedded in the source code, which adds to the code's complexity. At some point the precompiler directives may actually be viewed as a part of the language itself. The result is that several incarnations of the language coexist and platform neutrality becomes impossible. Porting the code becomes a nightmare.

The other approach involves breaking down the process of compilation in two stages. First, an abstract "platform" that the programming language compiles to is defined. This is not an actual platform, but rather an instruction set definition for a "virtual machine" (VM). The code written in the programming language always compiles to this VM, regardless of the actual hardware platform. The next stage of this approach is to translate the machine instructions of the VM to native hardware-specific assembler code. Figure 1 illustrates these two approaches for source code compilation.

This probably sounds like six of one and half a dozen of the other. And it is. The end result is the same – programming language code is translated (compiled) down to native machine code. The difference, however, is that the responsibility of configuring the compilation process is abstracted away from the programmer. Programmers write source code that always compiles to the same virtual machine. The virtual machine code is interpreted by a runtime VM that translates each virtual instruction into native code. The programming cycle becomes less complex, source code written and compiled on any platform runs on any other and the world is a happy place.

The Java Virtual Machine

The Java Virtual Machine (JVM) is an abstract machine that provides a specification for running compiled Java code. The specification allows flexibility in implementing the features of the VM. All JVM implementations need to support the execution of Java bytecodes, which are the virtual machine instructions that Java source code compiles to. However, the specification does not dictate how the bytecodes have to be executed. This functionality is provided by the vendor who provides the runtime environment. The VM may be implemented completely in software or, to varying degrees, in hardware. This flexibility allows the JVM to be implemented on a variety of computers and hardware devices.

The Java programming environment thus may be categorized into two computing environments. The compile-time environment provides the translation of Java source code to bytecodes (.class files). The runtime environment provides the interpretation of the bytecodes into native platform code. Figure 2 illustrates the role of the JVM.

The specifications of the JVM are officially released by Sun Microsystems. This means that the definition of the virtual machine instruction set is standardized by a single source. The specifications of the Java APIs are also officially released by Sun Microsystems. The JVM and the Java APIs constitute the Java Platform, or the Java runtime system.

As far as Java Virtual Machines go, we're not just in Cupertino anymore. While it's true that the official Java language definition is still provided by Sun Microsystems, alternate definitions of the JVM are creeping into the computing arena. Of these, the ones that have attracted the most attention are the Microsoft Virtual Machine for Java and the HP-Embedded Virtual Machine for Java. Both diverge from Sun's specifications for a Java Virtual Machine. Another JVM of interest for our discussion here is the long-awaited Sun Microsystems' HotSpot JVM.

This month the Cosmic Cup briefly examines the three JVMs mentioned above. While the first two "stray" from the specification, HotSpot adheres to it (after all, it is being developed by Sun), and illustrates a novel implementation of the JVM specifica-

Java Virtual Machine	Description
HotSpot JVM	Sun's next-generation virtual machine implementation that enhances Java's runtime performance
Microsoft VM for Java	An independently developed implementation targeted for Microsoft Windows platforms
HP-Embedded Virtual Machine for Java	An independently developed implementation targeted for embedded devices running real-time operating systems

Table 1

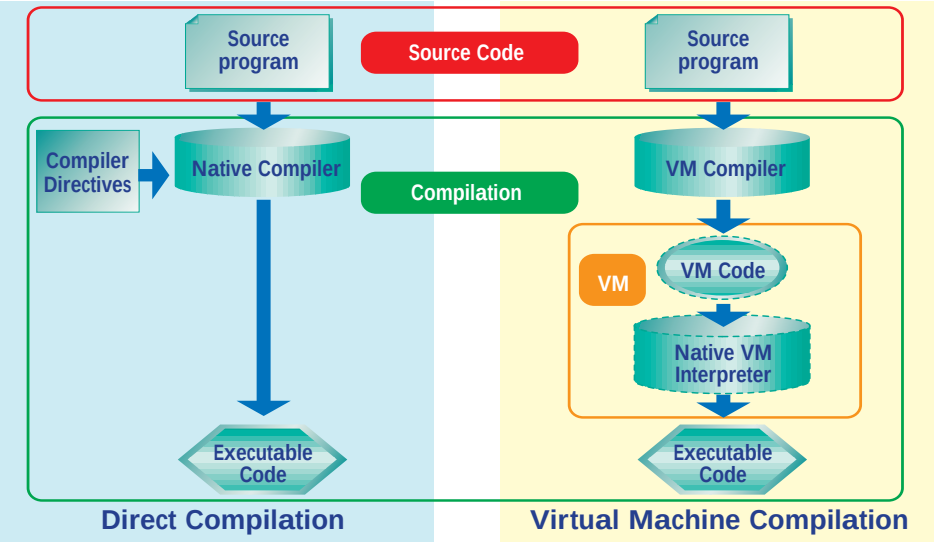


Figure 1: Direct compilation versus VM compilation

tion. These JVMs are shown in Table 1. They are discussed in the next section.

HotSpot Java Virtual Machine

The HotSpot JVM, an implementation of the standard Java Virtual Machine, is expected to provide significantly higher performance than other JVM implementations. It uses a variety of techniques to gain performance, including on-the-fly adaptive optimization technology, fast thread synchronization, a highly efficient garbage collector and an optimizing native code compiler. The key feature that makes HotSpot a novel implementation of the JVM specification is its use of modern dynamic compilation techniques to generate highly optimized code on the fly for specific execution environments. At the heart of the HotSpot JVM is the Java HotSpot Compiler. Unlike other compilers that compile the entire program before execution, the Java HotSpot VM runs the program immediately, using an interpreter, and analyzes it as it runs to detect the critical "hot spots" in the program.

The initial version of the Java HotSpot virtual machine has been designed to the JDK release version 1.2 specification, and is expected to be released at the same time as JDK 1.2.

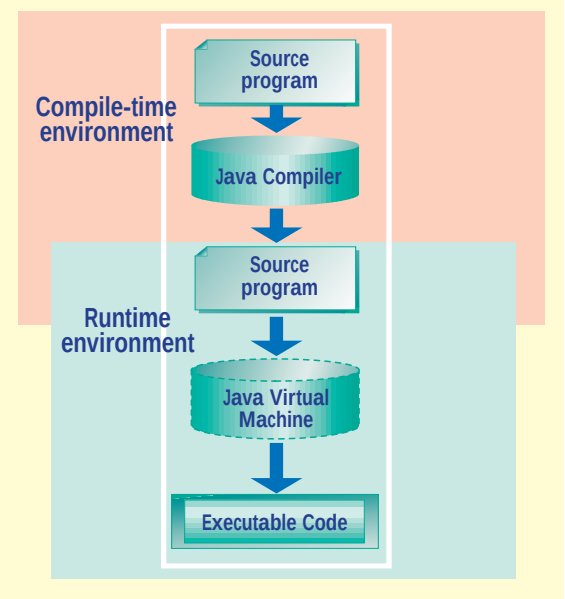


Figure 2: The role of the Java Virtual Machine

Microsoft VM for Java

The Microsoft VM for Java provides a complete implementation of Sun's JVM specification. However, it provides extensions to the specifications that target Windows platforms. The VM extends the standard JVM to provide the capability to load component object model (COM) classes and expose COM interfaces of Java classes. This means that the Microsoft VM allows Java classes to exist as both standard Java classes and ActiveX controls. The Microsoft VM itself is implemented as an ActiveX control.

This extension of the standard VM negates Java's platform independence. Once applications are developed using the extensions, they become specific to Windows platforms and won't run in other environments. Microsoft's VM is provided with Internet Explorer, Java SDK and Visual J++, Microsoft's IDE for Java.

In addition to the extensions to the VM, the Visual J++ environment also provides an extension feature to the Java programming language – a new keyword, *delegates*, and a supporting keyword, *multicast*. Since our discussion here is focused on JVMs, I won't go into a discussion on the construct itself. What it signifies in our context is that this small addition to the language means that the code using these keywords will also be compiled down to the JVM implementation. In other words, Microsoft VM will have to provide the code for compiling the keywords in addition to the standard Java language construct.

HP-Embedded Virtual Machine for Java

The Hewlett-Packard-Embedded Virtual Machine for Java includes a core set of Java API implementations for java.lang, java.net, java.util and java.io APIs. It specifically targets the embedded device market. It is portable to MIPS-4300, Motorola 68 K, Intel 80x86 and sARM processor-based devices.

This VM is not an implementation of a JVM specification provided by Sun Microsystems, but an implementation of a completely different VM specification defined by Hewlett-Packard. The specification for HP's VM was announced prior to Sun's Embedded Java specification (which is its counterpart). HP's Embedded Virtual Machine for Java provides incremental garbage collection and optional Java class libraries (i.e., all the libraries need not be linked at runtime), and has a very small footprint (approximately 0.5 KB). It is supported on HP-UX/PA-RISC and Windows NT/Intel development platforms.

The HP-Embedded Virtual Machine for Java is currently available as release 1.0.

Cosmic Reflections

In an ideal Java universe the definition of the computing environment would come from one source, all businesses would agree to specifications and standards proposed by one representative vendor, software releases for the JDKs would be on time and robust and there would be only one development and deployment platform. None of these fantasies come true in the real world, however. Different aspects of the computing environment are addressed by different vendors. Optimizing performance requires platform dependence by its very definition. And no one source can satisfy the needs of the entire industry. Since the problems that Java is targeting address the entire business community, it's unrealistic to expect the Java Platform to be generic and at the same time address every facet of the industry. As Java matures, we should expect to see other independent implementations of its VM.

About the Author

Ajit Sagar is on the technical staff at i2 Technologies, Dallas, Texas, and a Java-certified programmer with eight years of programming experience. He holds a BS in electrical engineering from BITS Pilani, India, and an MS in computer science from Mississippi State. He can be reached at Ajit_Sagar@i2.com.

Ajit_Sagar@i2.com


The Java Image Management Interface

The tightly integrated components of JIMI provide a powerful set of features in a modular and extensible architecture

by **Luke Gorrie & Michael Sick**

If a picture is worth a thousand words, what words do you use when the picture is in the wrong format? Not exactly a Zen koan, but a valid question whose answer is JIMI, the Java Image Management Interface from Activated Intelligence.

JIMI is a toolkit for reading, writing, viewing and manipulating images in multiple-graphics file formats. JIMI supports an impressive number of image formats, but if your target format is not supported, JIMI's open design allows you to add your format while taking advantage of key JIMI features. As you'd hope and expect, JIMI is platform-neutral 100% Pure Java that works with Java l.0.x and 1.1.x, and with the current beta versions of Java 1.2.

This article will let you know if JIMI has the features that will lighten your load and tighten your code. The "Overview" section examines JIMI's major features and identifies how JIMI can fit into your project. The "Down to Business" section focuses on technical topics and code samples.

Overview

When you finish this overview, you'll understand JIMI's key features and be able to identify how JIMI can provide a strategic advantage to your projects and products.

JIMI Encodes and Decodes Images

Life gets pretty frustrating when you run into communication barriers. The unruly universe of the World Wide Web, with its varied platforms, browsers, languages and users, can make a programmer cry out for simplicity. Java simplifies programming across varied platforms and environments. JIMI delivers stunning simplicity to the Java programmer who needs to work with the following formats:

GIF
JPEG
Portable Network Graphics Format (PNG)
Adobe Photoshop (PSD)
Targa (TGA)
Windows Bitmap (BMP)
OS/2 Bitmap (BMP)
ICO and CUR
Macintosh PICT (PCT)
Tag Image File Format (TIFF)
X Bitmap (XBM)
X Pixmap (XPM)
Sun Rasterfile (RAS)
PCX

The following code snippet reads a Photoshop file and writes it back to disk as a PNG file. It couldn't be easier!

```
Image image = Jimi.getImage("myImage.PSD");
Jimi.putImage(image, "myImage.PNG");
```

See Listing 1 for a complete and equally simple application version of this code.

Activated Intelligence's customers are using JIMI's simple encode/decode features to add serious power to their applications. Put JIMI in your applets and applications to easily browse and view most of the images that come your way. Place JIMI in your File Upload Servlet and rest assured that it'll accept a wide range of image files for conversion to popular viewing formats. Expect the number of supported formats to grow with each release.

Image Manipulation with JIMI

You should now be confident that you can read, write and view most image formats. JIMI helps you manipulate these images simply and uniformly. Going far beyond the AWT, JIMI gives you power over your images with pixel-level access. With this feature you can write new and standard image manipulation routines. Activated packed JIMI with a few favorites, such as:

Scaling (does thumbnails)
Cropping
Color-reduction
Smoothing
Flipping
Edge detection
Embossing
Tiling
Oil painting
Blur
Gamma adjustment
Smoothing

If you choose to write a custom routine using the JIMI pixel access interface, it'll work on all the supported formats.

What Makes JIMI Special?

If reading, writing, viewing and manipulating multiple image formats across multiple platforms isn't enough for you, JIMI has more. It's the foundation for imaging within the Java environment. What makes Activated so confident in JIMI? JIMI knows Java! JIMI protects the Java programmer from some vexing conditions in the Java environment. JIMI is ready to work with the Java community and harness its power. Following are four of JIMI's special attributes:

1. *Big images, little JVM:* How much memory will your Java application have at runtime? You may not know. JIMI has a fast, powerful and unique virtual memory management (VMM) system that lets you do more in your little JVM than you ever imagined. Using JIMI, a developer can load a 10000x10000 image in a JVM that is allocated only the standard 16 mb of heap space. JIMI uses memory only for the parts in use at that moment. How? JIMI leverages its fine-grained control over image access to implement a highly efficient virtual memory system tailored to the needs of large images. But the developer doesn't need to know the details to use the VMM. JIMI simply has the information you need buffered for fast access.
2. *Can I use JIMI on the server?* Yes. JIMI can work independently of the AWT. Use JIMI in servlets, Enterprise JavaBeans or any other AWT-less server-based environment. When you have lots of image translation or manipulation to do, JIMI lets you move the processing from the client to the server.
3. *JIMI is extensible!* Adding a new image format? No problem. JIMI's capabilities are dynamically extensible, allowing you to add new image formats. Each new format can seamlessly take advantage of all JIMI's core features.

   Because we designed JIMI to be easily and openly extended, Activated Intelligence is on the lookout for new image manipulation routines and image encoders/decoders. Developers can contact Activated Intelligence at www.activat-

ed.com to make arrangements for review and possible equitable use of their JIMI-based code.

4. *Flexible footprint:* With Activated Intelligence and independent developers adding new features and formats to the JIMI system, JIMI is growing! But what if you want to use it where memory is a constraint or the code needs to be moved over a slow network? JIMI is a modular system that allows the formats and features to "Plug and Play."

Down to Business: Programming with JIMI

Now that you know what JIMI can do for you, the rest of this article will show you how to become a JIMI developer. When you finish reading, you should know how to program with JIMI.

JIMI is made up of several tightly integrated components that provide a powerful set of features in a modular and extensible architecture. Figure 1 illustrates JIMI's internals.

The JIMI core provides the central functionality to support all other components. These classes handle the transparent AWT Image import/export, seamless VMM and other advanced imaging features. With a high-level understanding of the JIMI core, the developer can concentrate on the project.

The JIMI Image Read/Write Interface

The most fundamental use for JIMI is loading and saving images. JIMI's Image Read/Write Interface includes an intuitive front end with familiar and easy-to-use methods for image load-and-save operations. See how the "Jimi" class provides intuitive one-call methods for loading and saving images:

```
public static Image getImage(String filename);

public static Image putImage(Image image,
String filename)
    throws JimiException
```

Developers that have loaded images using the core Java API will be familiar with the getImage method. Jimi.getImage also provides asynchronous image loading, but with JIMI's added full range of format support, on-demand VMM and animated image support. Other convenient variations of getImage are provided, including a similar getImageProducer method used to apply filters or for use in an AWT-less environment. With these variations there's always a single method you can use to load an image to suit your needs.

Not surprisingly, the putImage methods are equally easy. Give JIMI the image (or ImageProducer), tell it where to save it and JIMI does the rest. Again, putImage has all the common variations to ensure that saving your image with JIMI is a one-call operation. Images saved will also automatically inherit any properties they were loaded with, such as compression scheme and interlacing options, even if you're saving to a different format.

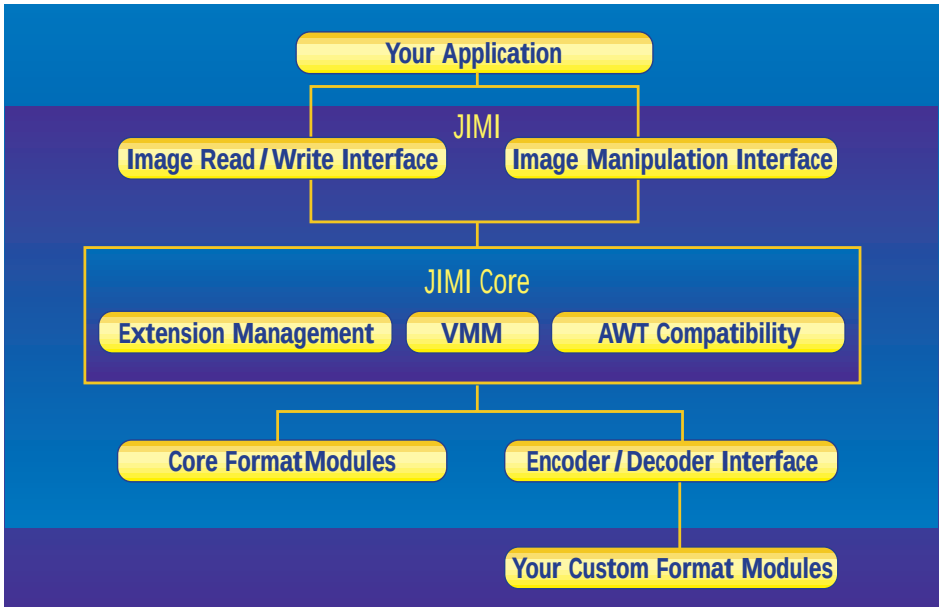


Figure 1

With these methods in your toolbox you have everything you need to quickly add basic load/save support for a diverse range of formats to your programs. JIMI takes things a step further by also enabling you to dissect and reassemble image files containing several frames of image data, such as animated GIFs. This is achieved with two new interfaces: JimiReader and JimiWriter. For more on working with multi-image files, see Listing 2.

See how easily JIMI's features enhance your programs? To get the JIMI advantage, just change the one getImage line in your code. Add one call to JIMI's "putImage" and the job is done.

The Image Manipulation Interface

Now you can easily load and save images, but how about modifying them? For this, JIMI has full compatibility with the AWT's ImageFilter model of image manipulation as well as compatibility with new JDK features like Java 2D. A common image-processing operation is applying an ImageFilter. With JIMI you can easily load an image, apply a filter and then save the image again. This code will crop a 100x100 region of an image and then save it:

```
ImageProducer
p1=Jimi.getImageProducer("i.gif");
ImageFilter f=new CropImage-
Filter(0,0,100,100);
ImageProducer p2=new FilteredImageSource(f,
p1);
Jimi.putImage(p2, "i.gif");
```

JIMI also provides more sophisticated image manipulation tools. Without JIMI, a developer who wants to access specific pieces of image data needs to use a PixelGrabber to get an image into an array of pixel data and then create a MemoryImageSource to build the modified image. This is expensive in both time and memory. JIMI provides random access to pixel data in its images. You have the power associated with an in-memory buffer and the flexibility that comes with all of JIMI's interfaces. Access your image in comfortable chunks and still benefit from JIMI's VMM capabilities.

Two additional key interfaces are JimiRasterImage and JimiBufferedRasterImage. The former provides a uniform interface to images created by JIMI regardless of their ColorModel type and form of primitive storage, letting you access their pixel data as RGB color values. The methods for this access are:

```
public void getRectangleRGB(int x, int y,
 int w, int h, int[] buffer,
 int offset, int scansize)

public void getRowRGB(int y, int[] buffer,
 int offset)

public int getPixelRGB(int x, int y)
```

JimiBufferedRasterImage extends this set of methods to include symmetrical "setters" for changing pixel data. The Jimi class provides a one-call method for accessing these objects. Just replace "getImage" with "getRasterImage." JIMI also provides buffered access to these

images with "getBufferedRasterImage" and the JimiBufferedRasterImage object. For more detail see Listing 3.

The Encoder/Decoder Interface

Now you can load, view, save and manipulate images for all JIMI's image formats, but what if your pet format isn't supported? Using JIMI's extension API you can add a new format module, leveraging all of JIMI's features and much of the work done for you. Implementing an encoder or decoder with JIMI gives you free image production, VMM support and seamless integration with the rest of JIMI.

Although greatly simplified, implementing format modules still deserves a more thorough guide than can be given in this article. Interested developers should take a look at www.activated.com/jimi/addformat/ for a step-by-step guide.

Conclusion

Now you know: JIMI's simple elegance lets you read, write, view and manipulate most images across multiple platforms in varied Java environments. You know when and where JIMI can help bring your product to release faster and with more features. You also know enough to get started writing powerful and effective imaging code using the JIMI toolkit. So go ahead, visit us at www.activated.com, download JIMI and "get the picture."

About the Author

Luke Gorrie is a Java programmer and the lead developer of JIMI at Activated Intelligence. He can be reached at luke@activated.com.

Michael Sick is a Java programmer, a Java advocate and the director of strategic development for Activated Intelligence. Contact him at mike@activated.com

luke@activated.com mike@activated.com

LISTING 1

```
Basic image conversion with JIMI

import com.activated.jimi.*;
import java.awt.Image;

public class JimiIsReallyEasy {
 public static void main(String[] args)
  throws JimiException {
  Image image = Jimi.getImage("myImage.PSD");
  Jimi.putImage(image, "myImage.PNG");
  System.exit(0);
 }
}
```

LISTING 2

```
Reading and writing multiple image files

import com.activated.jimi.*;

import java.awt.*;
import java.util.*;

/**
 * A simple example program for reversing the
 * order of frames in a multiframe image.
 */
public class ImageSeriesReverser
{
 public static void main(String[] args)
 {
  // print usage if wrong arguments are given
  if (args.length != 2) {
   System.err.println("Requires args: <src> <dest>");
   System.exit(1);
  }

  // vector to store loaded images in
  Vector images = new Vector();

  // read the images
  try {
   // create a JimiReader to read the image
   // series from
   JimiReader jr = Jimi.createJimiReader(args[0]);
   // enumerate the images in the series
   Enumeration e = jr.getImageEnumeration();
   // insert them into the vector in reverse
   while (e.hasMoreElements()) {
    Image i = (Image)e.nextElement();
    images.insertElementAt(e, 0);
   }
  }
  // catch exception if the source file
  // is malformed
  catch (JimiException e) {
   System.err.println("Error: " + e);
   System.exit(1);
  }
  // write the images back in reverse
  try {
   // create a JimiWriter for the output file
   JimiWriter jw = Jimi.createJimiWriter(args[1]);
   // pull the images out of the vector and
   // into an array
   Image[] series = new Image[images.size()];
   images.copyInto(series);
   // set the image array as the source for
   // the JimiWriter
   jw.setSource(series);
   // write the images!
   jw.putImage(args[1]);
  }
  // catch exception if the output file
  // can't be written to
  catch (JimiException e) {
   System.err.println("Error: " + e);
   System.exit(1);
  }
  // finished!
  System.exit(0);
 }
}
```

LISTING 3

```
Image Processing example method

public JimiBufferedRasterImage flipImage(JimiRasterImage source)
 throws ImageAccessException
{
 // the dimensions of the source image, which
 // are used to create a same-sized target image
 int width = source.getWidth();
 int height = source.getHeight();
 // create an empty target image to populate
 // with flipped data
 JimiBufferedRasterImage target =
  Jimi.createBufferedRasterImage(width,height);
 // a small buffer for pixel data to copy-between
 int[] rowBuffer = new int[width];
 // the index of the last row of the image
 int lastRow = height - 1;
 // loop through, copying each row from source to destination
 for (int row = 0; row < height; row++) {
  // get the source row
  source.getRowRGB(row, rowBuffer, 0);
  // copy it into the destination image at the
  // opposite location
  target.setRowRGB(lastRow-row, rowBuffer, 0);
 }
 // all finished, the target image is now fully populated
 return target;
}
```

Cyberflex Open 16K Development Kit

by Schlumberger

A development toolkit for building JavaCard 2.0 Applications

by **Jim Milbery**

Cyberflex Open 16K Development Kit

Schlumberger Smart Cards and Terminals

9800 Reisterstown Rd.

Owing Mills, MD 21117

Phone: 800 825-1155 ext. 208

Fax: 410 363-4336

Web: www.cyberflex.slb.com

E-mail: cyberflex@slb.com

Minimum Requirements: Windows 95/NT 4.0, 486 processor, 32 MB RAM, 50 MB disk space

Price: $499

One of the coolest pieces of technology I'd ever used was a programming interface for HK Systems' mechanical cranes. However, my coolness factor just went through the roof when I got a chance to try out the Cyberflex Open 16k Development Kit from Schlumberger Smart Cards and Systems. Schlumberger is well known in high-tech oilfield and measurement systems, and the Cyberflex development team is part of their Testing and Transactions group. The Java Card technology offers some exciting opportunities for building security and "wallet" applications using the Java language, and Schlumberger is at the forefront of this effort.

Product Installation

The Cyberflex kit ships with a Litronic 210 card reader, two Cyberflex smart cards and the development software on a CD-ROM. The installation process itself is straightforward; it's built with DemoShield so it behaves like a typical product installation. There are four pieces of software to install: the Microsoft Smart Card Base Components, the Litronic 210 Reader Driver, the Cyberflex Toolkit and the Cyberflex Documentation and Samples. Each installation requires a reboot operation, which slows down the overall installation, but it still took only ten minutes to install everything. The biggest challenge is getting the Litronic card reader plugged in and operating, as it uses a 25-pin serial adapter to connect to your workstation. Since I was running on a laptop, I was forced to dig into my bag of tricks and find a 25-pin to 9-pin converter (which Schlumberger is now including in their kits). The reader is powered off the keyboard connector, and I was able to connect my external mouse and the reader power supply into the same port without difficulty.

Developing a Java Card Application

I'd advise you to read through the Programmer's Guide and the Java Card 2.0 specification document before you try working with the software. The Java Card 2.0 API is based on the ISO 7816 framework for smart cards, and not all smart card vendors implement all features for all parts of the ISO 7816 standard. Although I don't have a background in smart-card programming, I could follow the manual fairly easily. There are several key terms to understand before you get rolling. The first is Application Protocol Data Unit (APDU), the message you send between your application and the smart card. The Card Acceptance Device (CAD) is the physical device into which the smart card is inserted. The Java Card Runtime Environment (JCRE) is the Java Card Virtual Machine and core Java Card API classes. You should also remember that, by definition, objects on the Java Card are persistent, i.e., values persist from one CAD session to the next.

Schlumberger provides a number of sample applications as Java source files along with the development kit, and I suggest you start with the examples before striking out on your own. While the Java Card technology is powerful, you have to get used to working in a 16 K environment again, and the command language for ISO 7816 can be somewhat limiting if you

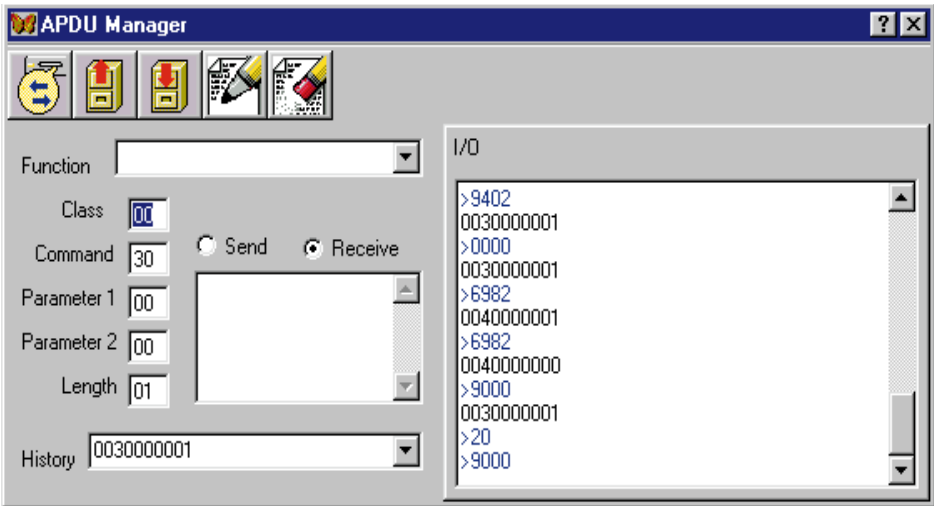


Figure 1: Wallet program

aren't used to it. The Java applications you build use only Boolean, byte and short data types; your code must be derived from the applet class and compiled with the debugging option set, even for production code.

I ignored convention and started with the most complex of the four sample applications, the Wallet application. I used Oracle's JDeveloper 1.0 to open the Java source code for Wallet and followed the instructions in the tutorial for compiling and testing the application. You'll find the instructions at this point a trifle unclear since you need to use the simulator class libraries to test your program and the tutorial doesn't explicitly point you toward that. In general, I found the printed documentation a bit skimpy, but the Cyberflex Web site (www.cyberflex.slb.com) has a good newsgroup bulletin board. Thanks to the numerous postings from Schlumberger's Anthony Chen, I quickly figured out that I had the wrong libraries. Once I fixed that problem, I used the APDU simulator to test the Wallet program (see Figure 1).

Deploying the Code

The APDU Manager, part of the Cyberflex toolkit, allows you to input commands and data to your program as if the code was running on the Java Card itself. On the left-hand side of the panel is the input window, which you can use to input a command string to your Java program. The right-hand panel of the form shows the output from the program after processing your command. The simulator offers some clever additions, such as the ability to save commands for later reuse. Once I successfully tested the program with the simulator, I used the MakeSolo utility to convert my Java class file into a stand-alone executable program and deployed the code to the Java Card without difficulty.

Building Your Own Program from Scratch

Once you're ready to begin developing programs from scratch, Schlumberger provides a simple, elegant skeleton Java program from which you can base your code. The skeleton code is derived from the Java applet class, and there is basically a five-step process to build your own program. First you create the applet name and class of instruction (CLA), then definitions for the primitive commands you'll be supporting. Next you define and allocate your instance variables for the application and create your class methods for the processing you plan to support in your program. The final step is to connect your methods to the APDU framework, which connects the messaging from the card to your application. Some restrictions are imposed on your code by the framework, and I'd suggest that you read the "Programming Requirements" section of the Developer's Guide before you start designing your own custom program.

Final Notes

Schlumberger is branding a complete line of smart technology under the Smart Village trademark, and they can print their Cyberflex cards with your company logo and information. The cards themselves have the same dimensions as a credit card with a small gold chip on the surface. Schlumberger's Cyberflex Open 16k Development Kit is worth looking into if you're considering using the technology for security or payment systems. Some other things you can get involved with are ID/personnel information, health care and telecommunications.

About the Author

Jim Milbery is an independent software consultant based in Easton, Pennsylvania. Jim can be reached at jmilbery@milbery.com or via his Web site at www.milbery.com.

www.milbery.com

FEEDBACK

"World Class" Awards?

"I Believe the Only Reason the Product Wins an Award Is They Pay SYS-CON for Advertising..."

Several months ago I noticed something in ***Java Developer's Journal:*** EVERY review published wins the World Class Award. or the Seal of Approval. I don't even read your product reviews anymore. There always seems to be an ad for the same product in your publication, which leads me to believe the only reason the product won the "award" was because they paid **SYS-CON** for advertising. I don't know this to be true, but it seems that way. I suggest that you indicate what criteria [are used to judge] products and how **SYS-CON** remains independent of the review.

I do, however, enjoy the articles you publish and have found many of the "how-to" articles very useful. Please keep up the good work on this front and good luck for the future.

Norm Cook
NWC Systems
cookn@cadvision.com

Thank you for your feedback. **Java Developer's Journal** *offers two of the most prestigious awards in the Java industry under the same strict guidelines we apply to our editorial content. These two awards are the* **JDJ** ***Readers' Choice Award*** *and the* **JDJ** ***Editor's Choice Award****. The first is determined solely by votes of* **JDJ** *readers;* **JDJ***'s editor-in-chief selects the second. Neither the awards nor the editorial direction of the magazine in general have anything to do with* **JDJ***'s advertising sales. In support of this statement,* **JDJ***'s advertising sales director has not communicated by phone, e-mail or any other means with* **JDJ***'s editor-in-chief for more than two years. We strongly believe that any publication with concerns for its integrity should be sensitive to the inherent ethical questions involved and impose similar measures.*

Your comments and observations regarding **JDJ** *product reviews are simply not true. Less than 30% of the products reviewed in the magazine are those of* **JDJ'***s advertising partners. You can verify this by examining the* **JDJ** *product review archives at our Web site.*

The criteria used to determine which products warrant a review for publication are simple. All new products submitted to us for a review are posted on our "Writers' Corner" Web site. Our writers then ask for the products to evaluate them. The reviews published with **JDJ***'s Seal of Approval reflect the quality and relevance of the products to our readership.* ***–Editor***

Can I Trust *JDJ*?

I considered taking ***JDJ*** up on the subscription offer for Java Lobby members, but I am afraid I must decline. A few issues back, ***JDJ*** came packaged with a free copy of MS J++. I consider this to be about as responsible as a desktop publishing magazine distributing a disc of Word viruses. I have not bought an issue of ***JDJ*** since.

Michael Cornall
mcornall@istar.ca

Java Developer's Journal*, as indicated in our mission statement over three years ago, is the premiere vendor neutral, technology driven, industry publication. We do not carry out a political agenda dictated by any of the vendors in the Java industry, including Sun Microsystems and Microsoft.* **JDJ***'s neutral policy is reflected in our editorial direction as well as our careful selection of articles and topics.* ***–Editor***

MICROSOFT'S RESPONSE

One of my favorite TV programs is a PBS series called *Connections*. In this show historian James Burke traces a number of seemingly unrelated technological advancements, inventions and scientific discoveries – through hundreds of years – and showed that in reality they are intimately connected and led to some of our modern marvels, such as man landing on the moon. One of the intriguing aspects of this show is that many of the inventions floundered, or outright failed, in the task to which they were originally applied. It was only after someone reapplied the technology that a breakthrough was made. We are masters of technology – we should be able to direct how it is best used to enhance our lives. The technology shouldn't necessarily constrain us to one course of action.

So it is with Java. There are those, like Mr. Cornall [see preceding letter], so focused on the philosophical issues they can't see the opportunities around them. Despite the hype, cross-platform is not the No.1 issue in the design of most systems. Further, the largest number of client machines are running Windows. For those situations where a "pure" Java developer needs to produce a compelling application for Windows, Visual J++ is the only logical choice. After all, an end user doesn't care one whit what language an application is written in. They want top-notch functionality, good performance and integration with other desktop software. Visual J++ is vastly superior to any "pure" Java implementation in this regard.

Are we alone in this view? Hardly. I'm sure I've shown you our tracking study data before, but let me refresh your memory. In our last survey (conducted via randomly dialed phone numbers until we complete approximately 700 developer surveys) we found that only 6% of developers mentioned Java when asked what languages they used. A recent InfoWorld survey found that only 4% of corporate developers had ever deployed a Java application. Finally, a recent ComputerWorld survey showed that 46% of Java developers used ODBC (a vastly Windows technology) as their DB connection and another 40% when through the JDBC/ODBC bridge (again a Windows technology).

Finally, if he had just focused on the content of your magazine I'm sure he would have seen you don't have an MS slant. Speaking of this, it sure would be nice to see some balanced editorial or technical articles on VJ6. It absolutely is the best Java development tool available.

Bill Dunlap
Visual J++ Product Manager
bdunlap@microsoft.com

"The Land of the Rising Sun" *by Alan Williamson*

There's a very interesting article in Volume 3, Issue 9 of ***Java Developer's Journal*** about a company that's been assembling a Web/Java-based system based on Oracle, all the problems they've been having with their driver, and their lack of support. Good article!

Alan Sparks
asparks@nss.harris.com
java-apache-users@list.working-dogs.com

Aha, a man who knows the nomenclature! The article you wrote was great; by the way, you're a very natural writer.

Bob Pasker
rbp@weblogic.com

I received my first issue of ***Java Developer's Journal***, Volume 3, Issue 8. Nearly every article previewed on the cover looked interesting and very useful.

The information in Alan Williamson's article about Oracle JDBC drivers, in particular, has the potential to save me many hours of frustration when I develop my first Java Oracle application.

I was also pleased to see the "full source code" for this article is available at ***JavaDevelopersJournal.com.***

All in all, I'm very pleased with my first issue and look forward to many more; I also look forward, hopefully, to better coordination between the physical and electronic editions.

Lee Levan
72030.3427@compuserve.com

CORBA 3.0 Update

Advanced features could make CORBA the way to go

by **JP Morgenthal**

In early September, to fill in the shadows left by industry rumors about the upcoming CORBA 3.0 release, OMG held a press conference outlining the details of this distributed-component environment. This month JP Morgenthal lays out the pieces and how they affect Java programmers.

Richard Soley
Editor, CORBACORNER
Chairman and CEO of the Object Management Group, Inc.

Currently hard at work, the Object Management Group (OMG) is preparing for a preproduction release of the CORBA 3.0 specification before year-end. Such a release will give CORBA ORB vendors an opportunity to implement new CORBA services and identify potential problems before the final release in the first half of 1999.

The preproduction release is essentially a test run of the specification, while the final specification will differ from the "P" release only in the changes necessary to correct problems.

A question arises about whether CORBA can continue to thrive in the face of all the emerging distributed computing technologies, such as Java Remote Method Invocation (RMI) and XML-based programming interfaces. The answer, unequivocally, is Yes. With the changes that will go into CORBA 3.0, not only will CORBA thrive, but it will finally be complete enough to support mainstream adoption and use by multiple levels of developers.

CORBA is primarily used by systems developers in the back office who have extensive knowledge of building distributed object applications. These developers are in high demand and are extremely expensive. But when it comes to integrating legacy COBOL applications with new C++ applications, CORBA rises to the top of the solution heap. That is, as long as a company can obtain and support the resources necessary to deliver that solution.

With 3.0, CORBA will achieve a new level of capabilities that will serve two groups: the hard-core distributed computing programmers and the less technical business programmer/analysts. Note that while CORBA 3.0 doesn't simplify distributed object computing, it does simplify the use of CORBA ORBs for the development of distributed object applications. Those building and deploying distributed applications must still understand the implications of networking, remote exception handling, object life cycles, multithreading and…the list goes on.

CORBA 2.2 and 2.3 Update

Before we can thoroughly dig into CORBA 3.0, we first have to look at the work delivered today by OMG-CORBA 2.2 and 2.3. Some important changes and new features being added to CORBA will impact CORBA users and the capabilities of the future 3.0 release. CORBA 2.2, which is available now, incorporated the following specification into CORBA:

- ***CORBA IDL to Java language mapping*** – The OMG formal specification for mapping CORBA types to Java objects.
- ***Portable ORB Adapter (POA)*** – This is an important initiative that will allow all future CORBA source code to be portable across ORB vendors. While CORBA will provide interoperability between ORBs, a server written for one ORB implementation is not necessarily portable to another. The POA provides a framework for managing object creation, policies and persistent identifiers, and a consistent service for objects that span multiple server lifetimes.
- ***COM/CORBA Interworking Part B*** – This specification identifies the interoperability of Microsoft DCOM and CORBA.

CORBA 2.3 simply amends the work of 2.2 for errors and problems and adds support POA for IDL/Java. POA for IDL/Java states that any CORBA server developed using Java and the POA shall be binary-interoperable. That is, any CORBA/Java application can be loaded into a compliant JVM running any vendor's ORB that supports the POA for IDL/Java.

CORBA 3.0 Features

CORBA 3.0 will bring many new features to CORBA that will facilitate its adoption in the enterprise. As previously noted, CORBA, even at the 2.3 level, is complex to implement. Some have identified Microsoft's COM architecture as simpler in some regards. But besides their both having activate and deactivate binary modules, that statement is like comparing apples to oranges.

The features listed and discussed below will be added to CORBA in 3.0. Where possible (some specifications haven't even received submissions yet), we'll identify the benefits a feature will bring to the enterprise.

- ***CORBA Component Model*** – Perhaps the most important and significant addition to CORBA, this model will specify a framework for the development of plug-and-play CORBA objects. It will encapsulate the creation, life cycle and events for a single object and allow clients to explore an object's capabilities, methods and events dynamically. The component model will significantly decrease the learning curve for developing and using CORBA servers and clients.
- ***CORBA Scripting Language*** – Scripting languages are significantly easier to learn than low-level programming languages. They remove complexities such as memory allocation and deallocation, memory pointers, and compilation and linking procedures. The CORBA scripting language will let client developers create and access CORBA servers while focusing on integration for the development of business logic.
- ***Objects-by-Value*** – CORBA has tradition-

ally passed references to objects around the network, which has simplified development of the ORB environment. References are merely aliases to the real object. Any methods calls made on the reference are deferred to the real object for processing. With the addition of objects-by-value, CORBA will pass the state of an object to another process for manipulation. When passing objects by this method, a proper type of object server must be available to handle the incoming object. However, as long as the object doesn't need to be activated, the object can exist in its "by-value" form, meaning it can be handed to a persistence service or a messaging service for further action.

- ***CORBA Interoperable Naming Service*** – This service is the facility that allows CORBA clients to look up and obtain a reference to a CORBA server. Since each ORB implements its own naming service, and naming services aren't readily interoperable across ORB implementations, a manual configuration is necessary to tell one naming service how to communicate with another. This specification identifies a way for independently developed CORBA clients to share a single naming context. The current submission has yet to define how bootstrapping – the process of naming services automatically detecting each other's presence – will be accomplished.

  Additionally, the specification adds support for URL-style naming conventions, though the current submission doesn't yet identify how this will take place.
- ***Multiple Interfaces*** – As submissions for this specification haven't been received yet, it's unclear what specifically this service will provide. Its intent, however, is to allow a single object to present multiple views of itself through an interface selection mechanism. If this sounds like Microsoft COM's interface query facility, one of the intended purposes of this specification was to more closely align CORBA with the COM object model.
- ***CORBA Persistent State Service (PSS)*** – Possibly one of the most controversial specifications for the OMG, this service will replace the CORBA Persistence Service, one of the least implemented services due to its ambiguity and complexity. CORBA PSS will provide a simple extensible interface that will allow CORBA to hide the implementation of storing and retrieving objects from the client and provide a consistent portable interface for CORBA server developers.

These six features embody the CORBA components initiative. Additional features include:

- ***Minimum CORBA*** – This specification identifies the requirement for obtaining the smallest footprint possible for a CORBA ORB. Specifically, this work will help jump-start the use of CORBA in embedded devices.
- ***Realtime CORBA 1.0*** – This feature will extend the CORBA specification for a new type of ORB called the Realtime ORB. It will expose interfaces so that developers will have more direct control over ORB resource allocation.
- ***Asynchronous Messaging*** – This specification has two components: different levels of quality-of-service agreements, and IDL changes necessary to support asynchronous method invocations. CORBA previously provided only three types of method invocation: synchronous, deferred synchronous and one-way. A one-way method invocation is a primitive form of asynchronous invocation that sends the message with no expectation for return. In the deferred synchronous method, control is returned to the client without a response, but the ORB synchronously invokes the method on the object and sends the response later. Thus the ORB still blocks while making the method call on behalf of the client as an agent.
- ***Asynchronous Method Invocation (AMI)*** – This defines a way for the client and the ORB to asynchronously invoke a method on an object. Hence the need for quality-of-service policies, which tell the ORB how to handle various delivery scenarios, such as when the object cannot be reached or requires too many jumps to deliver the method call. These quality-of-service agreements are implemented as a set of interfaces that manage policies, not a specific metric that must be met.
- ***Java Language to IDL Mapping*** – This facility will allow developers to build distributed applications completely in Java and then generate the CORBA IDL from the Java class files. Other binary applications can then access Java applications using RMI over IIOP.
- ***Firewall Support*** – The firewall specification defines interfaces for implementing IIOP through a firewall. It includes options for allowing the firewall to perform filtering and proxying on either side, which is important for extending the use of CORBA to the Internet and across organizational boundaries.

Conclusion

Simply speaking, the preproduction release of the CORBA 3.0 specification at year-end will provide ORB vendors an opportunity to correct problems that may arise during implementation of the new features and services. For many ORB vendors, some of these features may already be available either in a proprietary manner or one that is close to the released specification (this is often the case when the specification comes primarily from submitting vendors).

For organizations considering CORBA as a tool for integrating legacy and new systems, the new features and services could ease implementation considerably. In addition, companies may find CORBA to be among the most reliable of all interoperability software – an increasingly important issue in the age of distributed computing. However, the most likely to benefit from CORBA 3.0 are those attempting to address application integration across heterogeneous hardware and operating system platforms, where distributed computing resources are in short supply.

About the Author

JP Morgenthal is the principal analyst with NC.Focus, a rapidly growing analyst firm covering application integration technologies. JP can be reached via e-mail at JP@ncfocus.com.

JP@ncfocus.com

Bringing JINI Down to Earth

A way of "federating" virtual machines on a network to work together

by **Jason Rutherglen**

JINI is a new Java-based API from Sun Microsystems. Devices that are JINI enabled should be inherently able to talk to each other and even exchange code. JINI allows for store-bought devices to just "plug and work." What I'm going to discuss here is what JINI looks like to a Java programmer.

JINI comes out of the same group as RMI (Remote Method Invocation). It apparently used to be an acronym, but somehow lost it along the way. So a sort of prerequisite to programming in JINI is knowing some Java and being familiar with RMI. That said, let's begin. In JINI-land every device offers a service that's placed in a lookup for other devices to retrieve those services. Here's an example service:

```
public interface PrinterService extends
Remote {
  public void print(Object obj) throws Remo-
teException;
}

public class PrinterServiceImpl extends Uni-
castRemoteObject implements
PrinterService {

  public void print(Object obj) throws Remo-
teException {
    /** Code here to print the object **/
  }

}
```

The print method in this example is what other devices will call on the printer in order to print. So where's the JINI in all this? Glad you asked! We need to add a method to make the service register itself in the lookup (for the sake of this article we'll assume a lookup is running somewhere out there on the network). (See example below.)

```
public void register() {
  try {
    joinManager = new JoinManager(this,
                        entries,
                  new ServiceIDHandler(),
                        leaseManager);
  } catch (IOException ioe) {
    ioe.printStackTrace();
  }
}
```

JoinManager is a helpful utility class for registering a service with the lookup and managing the leases for you. When I first did some programming in JINI for a "whisper suite" demo for JavaOne '98, I didn't have this helpful class. Mine grew to be quite large. So now I've introduced the concept of a lease, which is basically a time-out renewal process so the lookup can discard old services that could be sitting in the lookup but are off the network (i.e., printer unplugged). With the JoinManager and a LeaseRenewalManager one doesn't have to worry about leasing. Here's the API spec for the constructor I'm using:

```
public JoinManager(java.lang.Object obj,
               Entry[] attrSets,
               ServiceIDListener callback,
               LeaseRenewalManager leaseMgr)
            throws java.io.IOException
```

As you can see, I haven't explained what an Entry is not and what a ServiceID is. A ServiceID is a unique identifier for each service. A service can already have a unique identifier or it can obtain one from the lookup. For the sake of our example we're assuming the printer doesn't have a ServiceID (although in reality an Epson JINI printer would probably have shipped with a ServiceID in the ROM). (See example below.)

```
public class ServiceIDHandler implements
ServiceIDListener {
  public class ServiceIDHandler implements
ServiceIDListener {
    public void serviceIDNotify(ServiceID
serviceID) {
      /** Save the service ID somewhere **/
    }
  }
}
```

Now we need to find out what "Entry" means. An Entry is an interface signifying that the object implementing it is an Entry (it has no methods!). When an object says it's an Entry, it's saying that it's serializable and used in JINI as a set of attributes of a service. That's why, when we're registering our printer service, the JoinManager constructor wants an array of Entries. One entry that's pretty easy to use is Name. So let's use it!

```
Entry[] entries = new Entry[1];
entries[0] = new Name("Epson Printer Ser-
vice");
```

We'd then pass this into the JoinManager. Assuming a lookup is on the network, we'd have registered our PrinterService remote interface in the lookup for communication with other services.

Let's discuss a few more high-level and interesting things about JINI. JINI is really a way of "federating" virtual machines on a network to work together, that is, JINI makes the network the computer. When most devices have JVMs, they all use the same instruction set. The network merely acts as the bus between two virtual microprocessors on a virtual motherboard. Also, JINI allows devices to distribute code among themselves. One practical example of this is the driver for the Epson printer. It can be downloaded by a JINI-enabled PC; all you have to do is plug in the printer and click print in the word processor and it will print – no installing drivers in Windows 95 or Windows NT Server, rebooting and all that stuff. Ultimately, JINI-enabled devices will allow for a new level of computing that is greatly simplified in comparison to the present state.

In a future article we'll build a JINI browser for browsing the contents of a lookup.

About the Author

Jason J. Rutherglen is CEO of the Silicon Valley startup RSDi, which provides distributed fault-tolerant e-commerce products and solutions on the various Java application servers. RSDi's flagship product is the Online E-Commerce Server. Jason can be reached at rsdi2000@yahoomail.com.

rsdi2000@yahoomail.com

Riding the Wave

Technological innovation sweeps the Java community once again

by **Don Preuninger**

As we approach the end of the second millennium, history teaches us that the only thing that's certain is change. Both natural history and human history consist of changes that occur in waves. Like ocean waves pounding the seashore, transforming rocks and cliffs into sand, so the waves of revolution and change have altered the course of human history time and time again. As a software development company, the ProtoView Development Corporation has experienced and ridden many waves of technological innovation, including the current wave of Java technology. Each wave brings with it new ideas and techniques to add to the picture – and at the same time removes things from the software landscape by way of making them unnecessary and obsolete. This happened previously with the mainframe in the '60s and '70s, the PC in the '80s, Windows, the Internet and now Java in the '90s. While these upheavals aren't pleasant, or in some cases even welcome, they are fundamental, necessary and, most of all, inevitable.

With the release of JDK 1.2 the Java community is once again being transformed by a towering wave of technological innovation. The single largest part of this wave is the Java Foundation Classes (JFC) library. This package consists of a series of fundamental classes and components that encapsulate just about every aspect of user-interface behavior. In effect, they replace the AWT library with a more sophisticated object hierarchy. From border types and color patterns to list controls and tables, the classes are designed to be for general purposes and reusable across many different applications and platforms. They even have the ability to morph themselves to conform to the expected appearance standards of the platform on which they're running. The JFC classes are superbly architected and seem to provide all the components and functionality one would want in a late twentieth century user interface. For the first time, the development community has at its disposal a standard set of UI objects that can be reused, extended and combined in countless ways to interact at many different levels.

> ***"Humans always find a way to build and improve on what exists in order to take it to the next level."***

So how should developers handle this new wave of technology? What does a company like ProtoView, which specializes in components of its own, do when a wave of technology looms and threatens to obliterate part of its own product line? The answer is simple: move to higher ground, welcome the wave's benefits and fish in it for all it's worth. Moving to higher ground means building new, more powerful, more sophisticated components on the base JFC components. By reducing the learning curve, which is considerable, and by combining and extending the capabilities of JFC, we're able to create an even more compelling technology for component-based development and code reuse. The base components can be extended to incorporate a simplified interface without sacrificing flexibility. They can be enhanced with features and behaviors that they're missing. They can also be combined into more powerful and sophisticated components that exceed the sum of the parts. For example, the ProtoView DataExplorerJ product integrates JTree and JTable components into a synchronized data exploring tool.

If many pundits of the '80s had been correct, most software developers would have been replaced by now with software robots and off-the-shelf packages, and the Java language itself would be unnecessary. Whatever wave comes along, there's always more to be done; humans always find a way to build and improve on what exists in order to take it to the next level. That's what ProtoView is doing and will continue to do with JFC and whatever the next wave brings. While it's certainly difficult to change and adapt to a fast-moving technology such as Java, at the same time it's essential that the development community adopt and absorb these innovations so that the Java language can mature as quickly as possible to everyone's benefit.

About the Author

Don Preuninger is vice president of research and development at ProtoView Development Corp. Don can be reached at donp@protoview.com.

donp@protoview.com

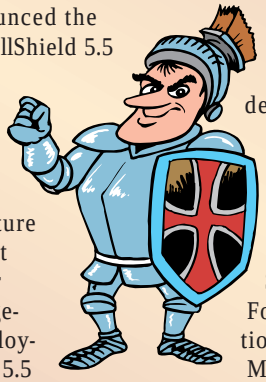
Oracle Chairman Invites JDJ to OpenWorld '98

(San Francisco, CA) – Chairman Larry Ellison of Oracle Corporation has invited ***JDJ*** to Oracle's OpenWorld '98, beginning November 7. In a personal note sent to Fuat Kircaali, publisher of ***Java Developer's Journal***, Ellison said he thought it would be "a lovely idea" to have ***JDJ***, the leading independent Java publication, present at the conference. OpenWorld '98, Oracle's annual user conference, is expected to attract over 15,000 attendees.

ObjectSpace Wins Place Among Top 100 on *Inc.* 500 List

(Dallas, TX) – ObjectSpace, Inc., has won a place among the top 100 on *Inc.*magazine's 1998 *Inc.* 500 list as a result of its exceptional growth rate. ObjectSpace is ranked 94th due to a 76% increase between 1996 and 1997. The *Inc.* 500 list is an annual ranking of the nation's fastest-growing private companies that have been in operation for at least five years.

InstallShield 5.5 Professional Rolls Out



InstallShield Software Corporation has announced the availability of InstallShield 5.5 Professional. This latest version of the company's installation development system includes a rich feature set of enhancement designed for easier enterprise and large-scale software deployment. InstallShield 5.5 Professional is designed to help application developers create powerful and reliable installations quickly and easily, as well as to assist network administrators to deploy enterprise-level Microsoft Windows applications.

InstallShield 5.5 Professional includes a number of feature enhancements designed to simplify installation for large-scale projects, and is available for a suggested retail price of $795.

For more information contact Michelle Kohler at 847 619-7044 or via e-mail at mkohler@installshield.com, or visit www.installshield.com.

KL Group Announces the Release of Olectra Chart 6.0

(Toronto, Ontario) – KL Group, a leading provider of GUI components and Java development tools, announced the release of Olectra Chart 6.0, the latest version of its charting tool for Windows developers. Olectra Chart 6.0 now provides developers with greater power and flexibility by combining high-performance graphing power with the freedom to create and customize virtually any type of chart.

Olectra ActiveX Controls

Olectra Chart 6.0 is an OLE/ActiveX control that offers a wide range of 2D and 3D charts and graphs specifically tailored to meet the needs of financial, technical and business users. For more information visit their Web site at www.klg.com.

Jinfonet Delivers JReport

(Washington DC) – Jinfonet Software, Inc., has announced the general availability of JReport 1.1, the first full-function report writer and server written totally in Java. JReport's 100% Java architecture means that JReport enjoys the full advantages of Java, including object orientation, portability, low administration costs and dynamic upgrades. JReport works with any platform, any database, any Web server and any browser. The product is available for immediate downlaod at Jinfonet's Web site.

JReport Workbench is priced at $995 per developer. The multithreaded JReport server is priced at $995 for up to five users, and an extra $195 for each additional user. A single-threaded version of JReport Server is also available at $100 per user.

For more information call 301 983-5865 or visit www.jinfonet.com.

Stingray Division of Rogue Wave Software Exhibits at PDC

(Denver, CO) – The Stingray division of Rogue Wave Software, Inc., exhibited its Visual Studio 6.0 support at Microsoft's Professional Developers Conference (PDC). Stingray demonstrated how its extensive line of design tools is tightly integrated with Microsoft's Visual C++ 6.0, Visual Basic 6.0 and Visual J++ 6.0.

In addition, Stingray demonstrated its MFC extension classes at PDC, the first ever integrated into the Visual C++ IDE. Stingray also presented its innovative frameworks in Objective Toolkit for WFC.

Stingray's recently released product line offers support for development on the latest Microsoft operating systems including Windows NT, Windows 98 and Windows CE.

For more information call Rogue Wave Software at 888 442-9694 or 303 473-9198, or visit www.roguewave.com.

ParaSoft Unveils New Tool Inuse

(Monrovia, CA) – ParaSoft has announced the availability of a stand-alone version of their development tool Inuse. Inuse helps developers optimize application performance by analyzing and graphically animating dynamic memory allocation in real time and reporting precise information about each potential problem.

Inuse has been available as an add-on to Insure++, ParaSoft's award-winning runtime error-detection tool, and can be run on any Windows executable. The executables can be built with any compiler and can be written in any language that compiles to an .exe file.

For more information call 888 305-0041 or visit www.parasoft.com.

OneRealm Focuses on Software Internationalization

(Boulder – CO) OneRealm, Inc., has been formed to provide products that drastically reduce the time to market and cost of preparing both Internet-based and traditional software products for global deployment. The company is focused on automating software internationalization, resulting in a quicker, more cost-effective way to make software ready for foreign markets. In addition, OneRealm enables companies not currently globalizing their software to reap the rewards of the larger international market.

For more information call the company at 303 247-1284 or visit www.onerealm.com.

ProtoView Releases the JFCDataCalendar

(Cranbury, NJ) – ProtoView has announced the release of the JFCDataCalendar, a high-powered, lightweight Java calendaring component to be bundled with the ProtoView JFCSuite.

JFCDataCalendar is built on the foundation of JFC(SWING) and includes features such as drop-down date selection, multiple date and range selection, and day-specific font/color/image properties.

For more information contact Noah Kuttler at 609 655-5000 (x118) or visit www.protoview.com.

SNiFF Available for SCO Openserver and UnixWare

(Santa Cruz, CA) – SCO and TakeFive Software have announced an agreement to make SNiFF+ v3.0.1 available for the UnixWare 7 and SCO Openserver Release 5 operating systems. Available through the SCO Developer Program, UnixWare and OpenServer developer kits will feature a trial version of SNiFF+ v3.0.1 that provides developers with the option to purchase SNiFF+ at a special introductory price through January 31, 1999.

By purchasing the kits, developers also have the option to purchase a single-user, Windows node-locked license for SNiFF v3.0.1 for $995 per license.

For more information visit www.takefive.com or www.sco.com.

IBM Simplifies Web Application Management

(New York, NY) – IBM has announced eNetwork On-Demand Server, software that makes it easier to deploy and use Web-based applications within an enterprise. As more businesses take advantage of Java technology, they need to ensure enterprise-class flexibility, reliability, security and ease of administration, as well as contain costs.

IBM eNetwork On-Demand Server does this by allowing customers to create "smart" Web applications that let them tailor application preferences down to the individual user, allowing greater flexibility in deploying and managing applications from anywhere in a network.

A prototype version of On-Demand Server will be demonstrated at the Java Business Expo from December 7 to 10, at the Jacob K. Javits Center in New York City.

IBM's Snehal Parikh, Steve Ims and Brooke Upton met with ***JDJ*** publisher Fuat Kircaali and editor Brian Christensen at ***JDJ*** offices to announce the product upgrade.

For more information visit www.software.ibm.com.

VisualWorks Product Line Expands

(Irvine, CA) – ObjectShare, Inc., has announced the availability of VisualWorks RoseLink 1.0 software that provides support for Rational's enterprise analysis and design tool, Rose98.

VisualWorks RoseLink adds modeling capabilities to the VisualWorks development environment. Developers are able to:

- Reverse-engineer existing projects building a blueprint of their applications
- Forward engineer Smalltalk code from a Rose model
- Port existing VisualAge Smalltalk applications to VisualWorks
- Document existing applications to build understanding of the application among existing and new members of their development team
- Define domain models for multilanguage applications

For more information call 800 759-7272 or visit www.objectshare.com.

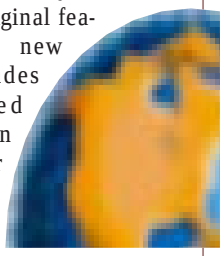
WhiteBox 3.0 Tool from Reliable

(Sterling, VA) – Reliable Software Technologies Corporation (RST) announced WhiteBox 3.0. This advanced code coverage tool enables software development and testing organizations to reap the benefits of software testing without changing existing development or testing processes. As a result, WhiteBox 3.0 improves productivity and time to market by optimizing testing resources.

For more information visit www.rstcorp.com.

New ILOG JViews 2.0 Delivers for Java GUIs

(Mountain View, CA) – ILOG has introduced ILOG JViews 2.0, an enhanced version of its award-winning ILOG Jviews 100% Pure Java graphics library. In addition to the original features, the new release includes sophisticated visualization features never before available in a single Java product – Web-based mapping, relationship displays and easy-to-design data-aware icons. ILOG JViews 2.0 is built to give developers the fastest, most comprehensive solution for designing and maintaining high-end, Web-based GUIs.

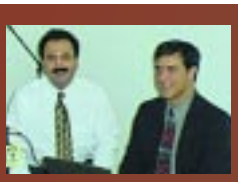


ILOG JViews 2.0 is available now, and prices start at $6,500. For more information visit their Web site at www.ilog.com.

Object Insight, Inc., Announces JaVISION

(Ann Arbor, MI) – Object Insight's JaVISION is a programmer's development tool that automatically generates UML Object Model Diagrams from .java or .class files, enabling an iterative code/design lifecycle. Teams can generate diagrams of their code and use the "Add Related Classes" or "Show Implementors" comand in JaVISION to show a class context, e.g., using classes from Sun's JDK as collaborators. Developers can open a source code editor for any class in a diagram, compile changes and run an applet from the diagram view.

JaVISION is available for Windows 95/NT as a 30-day "try before you buy" free software download at www.object-insight.com. The product can be unlocked with a $99 purchase at any time by the user.



Ron Suarez (r.), president and founder of Object Insight, visited the *JDJ* offices to announce JaVISION

For more information e-mail Object Insight at info@object-insight.com.

THE GRIND

by **Java George**

Application Servers: *Part 2*

Using Adapters to Go After Legacy Data

Legacy systems are too important for Java developers to ignore, but needn't be a hassle to access. These systems include everything from packaged applications like SAP and Peoplesoft to aging home-grown mainframe systems and external data sources such as EDI. Despite what you may personally think about some of the more elderly of these systems, they play critical roles in many organizations and will continue to for years to come.

In this column, the second of three in the application server series, we'll look at how developers handle legacy systems. Previously we discussed how developers accessed SQL results sets through the application server.

As developers we frequently want to bring a result set from the legacy system to the application server where we make it available for reading and updating. In fact, we may want to integrate any data into the application server as a result set. For instance, Java developers may have to incorporate data from EDI or, more recently, XML, the new, increasingly popular standard for structured data exchange over the Web.

XML (Extensible Markup Language) is particularly interesting. It is a tag-based markup language like HTML except that it can describe not only how data is to be displayed on the Web, intranet or extranet, but what the data actually represents. Once the computer knows what a piece of data represents – phone number, part number or customer name – it can programmatically process the data, store it and display it. XML is referred to as open and extensible because users can define the tags they need. Expect to find more and more legacy data increasingly taking the form of XML pages.

Some application servers handle legacy data through the use of adapters. The adapter acts as middleware, taking a standard call from the client and transforming it into a call to the appropriate legacy system API. It then presents the returning data in a manner that closely resembles a SQL result set, with rows and columns, so that the application server framework can easily work with the data. In addition to making API calls, such as calls to SAP's BAPI interface, adapters also read flat files, interpret EDI and XML uploads, and interface with application integration servers such as Active Software's ActiveWeb.

Application server adapters are relatively new to the market, but I expect they will become quite popular. Some application-server vendors provide them. Also watch for third-party providers to develop adapters for some of the more esoteric legacy applications to run with the more popular application servers.

Without commercial adapters, Java developers have to create their own. Those who have done it generally don't wish to repeat the experience. Creating an adapter requires handcrafting custom, low-level connections to the legacy application, a difficult task that generally produces a mountain of specialized code.

Complicating the task is the need to make the results usable. Once the data is retrieved, it needs to be cached, delivered, displayed, navigated and manipulated. Hand programming these on a data-source by data-source basis can be extremely time consuming and error-prone. In addition, every time the legacy application vendor enhances the application or changes something, the Java programmer must rewrite the adapter. I can think of better ways to have fun.

If you're a Java programmer just trying to knock out an application that requires a connection to a legacy system or packaged application, there is no glory in creating these adapters yourself. Shop around for an application server with built-in adapter support that provides the legacy connections you need and can integrate into your development framework. The application server's ability to incorporate such legacy data into a new Web application can provide tremendous benefits.

"Creating an adapter requires handcrafting custom, low-level connections to the legacy application, a difficult task that generally produces a mountain of specialized code."

Java George is George Kassabgi, director of developer relations for Progress Software's Apptivity Product Unit. You can e-mail him at george@apptivity.com.

George@sys-con.com