JAVA

SPECIAL ISSUE

JavaDevelopersJournal.com

Volume: 4 Issue: 1, 1999

Best Java Products of the Year

SPECIAL AWARDS ISSUE!
BONUS FREE AWARD WINNING JBUILDER 2.0 CD!

CELEBRATING JAVA DEVELOPER'S JOURNAL 4th Year!

A Close Look:
Over 20 Award Winning Java Products…

INSIDE THIS ISSUE:
Your personal access code to *JDJ's Digital Edition*
Read the most recent issues FREE!

Event Management & Enterprise JavaBeans:
Make it Easy to Write Your Java Applications

Design Patterns in a Java Interpreter:
They Can Aid in the Design of Complex Software

Tune into SYS-CON's Java Radio:
Over 50 Exclusive Live Interviews
from the Java Business Expo

SYS-CON PUBLICATIONS

SPECIAL EDITOR'S AWARDS ISSUE

JAVA™ DEVELOPER'S JOURNAL

JavaDevelopersJournal.com

Volume: 4 Issue: 1, 1999

Your Personal Acces Code to JDJ's Digital Edition: JAN9901

SYS-CON PUBLICATIONS

JAVA DEVELOPER'S JOURNAL

EDITORIAL ADVISORY BOARD

Ted Coombs, Bill Dunlap, David Gee, Michel Gerin, Arthur van Hoff, Brian Maso, John Olson, George Paolini, Kim Polese, Sean Rhody, Rick Ross, Richard Soley

Editor-in-Chief: Sean Rhody
Art Director: Jim Morgan
Executive Editor: Scott Davison
Managing Editor: Hollis K. Osher
Senior Editor: M'lou Pinkham
Production Editor: Brian Christensen
Technical Editor: Bahadir Karuv
Visual J++ Editor: Ed Zebrowski
Visual Café Pro Editor: Alan Williamson
Product Review Editor: Jim Mathis

WRITERS IN THIS ISSUE

Gene Callahan, Brian Clark, Claude Duguay, Anil Hemrajani, Rob High, Ken Jenks, George Kassabgi, Jim Milbery, Sean Rhody, Ajit Sagar, Bhaven Shah, E Ming Tan, Alan Williamson, Brian Zimbelman

SUBSCRIPTIONS

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

Subscription Hotline: 800 513-7111

Cover Price: $4.99/issue
Domestic: $49/yr. (12 issues) *Canada/Mexico:* $69/yr.
Overseas: Basic subscription price plus airmail postage (U.S. Banks or Money Orders). *Back Issues:* $12 each

Publisher, President and CEO: Fuat A. Kircaali
Vice President, Production: Jim Morgan
Vice President, Marketing: Carmen Gonzalez
Advertising Assistants: Robyn Forma, Jaclyn Redmond
Accounting: Ignacio Arellano
Graphic Designers: Robin Groves, Alex Botero
Webmaster: Robert Diamond
Customer Service: Sian O'Gorman, Paula Horowitz
Online Customer Service: Mitchell Low

EDITORIAL OFFICES

SYS-CON Publications, Inc.
39 E. Central Ave., Pearl River, NY 10965
Telephone: 914 735-7300 Fax: 914 735-6547
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

SYS-CON PUBLICATIONS

FROM THE EDITOR

Sean Rhody, Editor-in-Chief

New Year's Resolution

Welcome to 1999. This is typically the time I make predictions about the coming year. Next year I'll get this issue out and have a good laugh at all the things I missed – and the few I actually get right. I'd be remiss in my editorial duty if I didn't make some predictions.

This promises to be an interesting and tumultuous year. By the time you read this, there will be fewer than 365 days left to fix Year 2000 issues. In most cases, if you haven't already started fixing your applications, you won't finish. How this applies to us as Java programmers is less clear. In many cases OUR code may be fine, but if it relies on other systems or interfaces, or even legacy databases, we may still have time bombs on our hands. I wish I could say confidently that we've all addressed these issues, but reports indicate that the majority of the industry is behind schedule in remediation and repair.

Look for this to be a banner year for the EJB camp. I know of over a dozen companies, including all the major vendors, that are rushing to complete an EJB server. The EJB specification is deliberately vague on a number of issues, so look for vendors to try to differentiate themselves by adding value with different services and approaches. Some vendors are taking a pure Java approach, while others are uniting CORBA and EJB in a server that is implemented in native code, rather than riding on top of a JVM. Everyone but Microsoft will have an EJB server – Oracle's even putting it directly into their database.

Don't expect large-scale production applications using EJB until the end of the century. (I love being able to say that!) There's only one production server I know of as I write this – by the time you read this there should be five or six. Given the lead time required to tool up and actually write a large application, it will be the third or fourth quarter before we see a proliferation of EJB apps.

Sun has won the first battle in the lawsuit with Microsoft. Although Microsoft could yank support for Java in their browser, I expect that they'll appeal the ruling and delay changing their implementation for as long as possible. Though I'm not a lawyer, it seems to me that Sun's case is pretty strong, so I expect that Microsoft will eventually replace the VM and get on with business. Anything else would drastically affect the usefulness of the browser.

JINI will make small inroads this year, waiting for hardware and other vendors to catch their collective Java breaths. Standards have been coming fast and furious from Javasoft, but it takes a while…and a certain industry will…to make a standard more than a document. JINI looks good, but there's a certain amount of inertia with such standards until sufficient mass builds around it. I don't think JINI will take off this year.

Version 1.2 will become the main standard, supplanting 1.1.x. I see this as a year-end reality, again waiting mainly for the browser vendors to come up to speed, particularly with the new security model. The sandbox approach for applets was an easy out for Microsoft and Netscape; implementing a real security model will require a greater amount of effort on their parts. Look to Microsoft to have the first implementation, and to try to corrupt the standard yet again by making the security system interact with NT Domains. That's completely my prediction; but given their current tack, it's something I think is likely to happen.

We've got an event-filled editorial schedule for this year – this month we're focusing on JavaBeans and tools for GUI development, in March we're looking at middle-tier servers and in May we'll be focusing on hot new Java technologies. Visit our Web site at www.sys-con.com for a more detailed look at our calendar and other ***JDJ*** daily features.

Finally, look for a revival of The Artist Formerly Known as Prince, riding a retro wave of century madness on the basis of his song "Tonight We're Gonna Party Like It's 1999." I can already see the bad commercials and advertisements starting. Happy New Year, and party on.

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.*

EVENT Management & Enterprise JavaBeans PART 1

Making It Easy to Write Applications

by **Brian Zimbelman**

This is the first in a two-part series on Event management in large distributed applications built on top of Enterprise JavaBeans (EJB). This installment will cover the architecture and the implementation of a local (single VM) version of the event distribution system. The second article will implement an EJB version of the system that will handle distributed events from remote VMs.

One of the goals of Enterprise JavaBeans is to make it easy to write applications. Application developers won't have to understand low-level transaction and state management details, multithreading, resource pooling and other complex low-level APIs. In the nominal case this extrapolation of the complexities of building large, complex distributed applications has been accomplished well. It doesn't take much more effort to implement an interface to a database table object using EJB than it does to implement it locally. If the EJB object is deployed using a high-end EJB server, the object can be replicated among multiple machines, providing load balancing and fault tolerance with little or no effort on the bean developer's part.

The model that EJB is predicated on, transaction-based processing, works well in many circumstances, the most prominent being client/server and *n*-tiered systems in which the server layer manages some form of persistent store. A number of other distributed application needs, however, don't fit as well into the EJB model. One of them is event-based processing.

EJB still provides many benefits to applications that don't directly match the transaction-based EJB model. It just takes a bit more understanding of what EJB provides for you and what you have to do to get it to work. Event processing happens to be an application that begs for a middleware server but doesn't fall into the transactional model.

Events happen on a regular basis in our daily lives. We get up, go to work and so on. The same can be said of our software – events happen regularly in our software systems. The user logs into the application, requests a new client's data, etc. Some events, such as handling an order, are handled in the regular course of processing; others, such as an I/O failure, are considered exceptional processing.

In large distributed systems a frequent problem is that events often need to be processed by a component that resides far from the component that detected the event. Maybe the middle tier detected that the logon attempt failed, but a logging component on the server needs to know about this failure. On the other hand, the component that detected the problem shouldn't need to know what components are interested in this specific event. For distributed applications the event distributor needs to handle both the receiving and the transmitting of events to remote components.

In reviewing the strengths and weaknesses of EJB and how they fit in an event distribution subsystem, we can determine the following:

- EJB allows remote components to send events to the event distribution system.
- EJB doesn't provide a mechanism for the event distribution system to inform the remote components that an event has occurred (no callback mechanism).
- The EJB server provides all the features required for a highly scalable, fault-tolerant, load-balanced application.
- EJB's transactional model doesn't help us in the event distribution subsystem, since events are not transactional in nature.

This article and its sequel will show you one possible way to handle these vast requirements in a simple, straightforward framework so that your application will be able to handle events as easily as it handles normal, transaction-oriented, multitiered processing.

What Is an Event, Anyway?

In most Java development circles, when event handling is discussed it's normally in association with the AWT event model. In the AWT model the component interested in the event (consumer) simply registers with the producer of events (producer). The producer then iterates its list of interested parties (consumers), sending the event to all who have registered with it. This is a classic implementation of the observer pattern, and it works well in an application that runs in a single VM.

When an application begins to span multiple VMs, and components are created on the fly without notification to other components, the AWT model of tight coupling between the consumer of events and the producer of events gets to be unworkable. How does the consumer know that a producer of events has started on a machine somewhere else on the network?

Consumers of events don't care who generated the event, just that the event occurred. If a database is out of space, the event consumer doesn't care what process detected that the database needs more disk space. It just knows that its job is to notify the administrator of this fact. Conversely, the event producer doesn't care who needs the event, or what they're going to do with it, just that the event was detected and that consumers may be interested in it.

The mediator pattern can be used to alleviate the tight coupling in the basic event distribution model described above. To alleviate the tight coupling, a third component – the event distributor – is added to act as a mediator between the publisher and the consumer. The producer of events publishes the events to the distributor and thus doesn't know what objects are consuming the events. It just knows that they have been handed off to the proper component.

Consumers of events, on the other hand, are notified when events they're interested in are received by the event distributor. The consumers don't know what object produced the event, just that it occurred.

Thus we have three roles that exist in an event distribution system:

1. **Event Producers:** Components that detect events and so inform the world
2. **Event Consumers:** Components that handle events in some manner, e.g., passing them on to other consumers, notifying other applications or triggering some processing in the application
3. **Event Distributors:** Distributors that maintain a list of interested consumers and pass the events to the consumers as they are produced

By chaining event consumers together, an application manages the consumption of complex event hierarchies. Figure 1 shows the major components of the event distribution system. The three listed above are there, as well as the Event object itself.

Class Diagram

Figure 2 shows the class diagram for the event distribution system implemented in EJB. Notice that EventDistributor and EventConsumer are interfaces and the EventProducer doesn't even exist! Anyone can produce an event just by creating a new instance of an Event object and calling its publish method. The EventFilter class is an abstract class that implements both EventConsumer and EventDistributor.

The normal mechanism for creating an event consumer is to extend the EventFilter class, providing a consume method that performs some filtration on the events that are passed on to the consumers that register with it.

The Event, EventContext, EventConsumer, EventDistributor and EventFilter classes are the basic building blocks for local event distribution. In this article I'll implement these classes and their interfaces. The second article will explain the implementation of the rest of the classes diagrammed in Figure 2. However, for completeness, I'll discuss all of the classes here.

Enterprise JavaBeans

The distribution mechanism will be built on top of the classes in EJB. The EJB implementation of the event distributor is modeled as the EventController interface, which extends the EventDistributor interface and also provides the publish service so clients from any VM can inform the event distribution service of new events.

The EventController is actually an interface as well as a class on the client machine (generated by the EJB compiler). The EventController interface is implemented on the server by the EventControllerBean (as well as the EventControllerHome). With these three classes and one interface, EJB can provide the global interface to all clients who wish to publish events or consume events that meet certain criteria.

As I mentioned before, EJB has a severe limitation in the area of callbacks, so I've designed the class RemoteEventConsumer, which encapsulates the logic needed to provide a callback for the server when it needs to notify a consumer in another VM of the event. The last class shown in Figure 2 is the EventHelper class, which is used as a holder of two static methods that make the application programmer's job much easier.

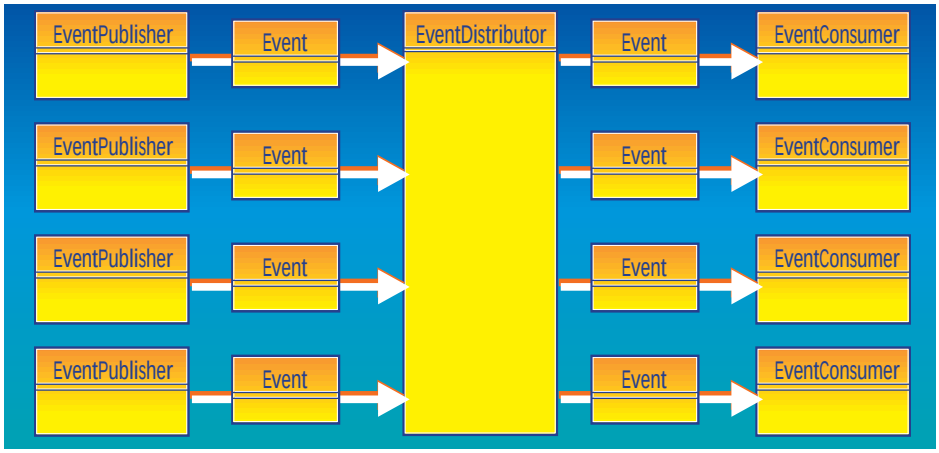


Figure 1: Event distributor

I'm not showing the relationship of Event to the other classes in the diagram as I don't believe it adds anything and it certainly clutters up the rest of the details.

Event Contexts

To delineate between different event types, one can create many subclasses of Event, or a type attribute can be added to the Event object. Implementing a large number of classes just to identify types of events is a design that causes application bloat. However, if different events have a vastly different state as well as behavior, they are best implemented as a class hierarchy.

The design doesn't require either method to be implemented, and you can determine which method is best for your application. The sample implementation that I'll go over later will use an EventContext object to identify the event type. It will also use a dynamic set of attributes (implemented as a hashtable) to contain whatever data is required for this type of event. While this implementation meets my application's needs, yours may vary.

Generally, events will exist within a context. For example, an invalid logon attempt is a specific type of security violation. This design requires that when a consumer is added to the event distribution system, its context has to be passed as an argument to the addEventConsumer method. For the example code I'll model the context in a string representation based loosely on URLs. Your application's needs may vary and you'll want to change how the context is represented. The URL approach works well for my needs as it's universally recognized and easy to parse.

The event context hierarchy will match the set of event types produced in your application. For my application a subset of the tree looks like this:

Security
Security.Breach
Security.Breach.InvalidLogin
Security.IllegalAccessAttempt

Resource
Resource.Database
Resource.Database.Corrupt
Resource.Database.InvalidAttempt
Resource.Database.NotFound

Chart
Chart.Update

Inbasket
Inbasket.NewMessage
Inbasket.DeleteMessage

Notice that the first two categories of events (Security and Resource) are generally considered errors and would for the most part be sent from the producer to some sort of administration tool (logging device, paging device, etc.). However, the last two types of events (Chart and Inbasket) are notification from the server to the client that some interesting data has been updated in some manner.

The EventContext class is constructed by passing in a string containing one of these URLs; it provides all the functionality needed for the rest of the application to obtain all or some part of the context. This allows the rest of the application to be abstracted from the need to understand and parse event contexts.

Why This Design?

As with any design, there are advantages and disadvantages. Among the advantages:

- It's simple. Simple solutions make the design easier to understand, implement and maintain.
- It takes full advantage of EJB server strengths. It allows the EJB server to replicate the object and provide the fault tolerance, load balancing and other high-end server functionality.
- It decouples the consumer from the producer. When two components are tightly coupled, the system is considered frail and brittle. Frail systems are hard to extend, and make maintenance difficult at best – impossible at worst.

The most obvious disadvantage of this design is the central distributor component. Any design that has a single point of failure (as this design appears to have) is suspect. However, with the capabilities of EJB servers, this single point of failure is easily mitigated. The EJB server itself can replicate the distributor, thus reducing the risk of a single point of failure.

Two weaknesses in standard EJB are exposed in this design:

1. It doesn't handle nontransaction-based applications well (which is to be expected of a transaction-based standard).
2. It doesn't provide callback functionality by default.

In EJB parlance, one would remedy these weaknesses by building a custom EJB container that fits into the EJB server of choice. This article chooses another direction, that of working within the existing containers of the EJB Server and providing the extra functionality in the bean object itself.

Proposed Solution

Enough talk. Let's write the code and see how it works. For the rest of this article I'm going to describe how the base classes needed for event distribution in this model work. (I won't discuss the networking or EJB aspects of these classes until the next article.)

Interfaces

Two interfaces, EventConsumer and EventDistributor, need to be implemented first. Listing 1 shows how the EventConsumer interface requires the consume method. An important note is that an EventConsumer extends the EventListener interface of Java.

The EventListener interface, a little-used interface provided in the java.util package, doesn't implement any methods. Rather, it tags the interface or class as one that can be used by an EventListenerList derivative. The EventListenerList class, provided in the com.sun.java.swing.event package, contains all of the mechanisms needed to main-

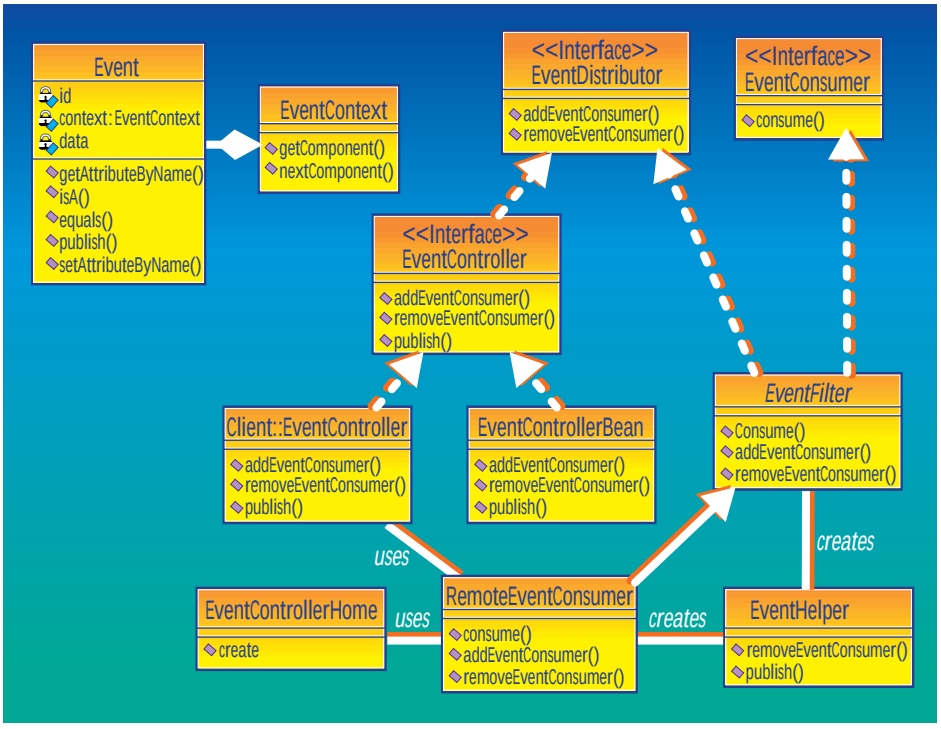


Figure 2: Event object model

tain and process callback lists.

Listing 2 has the code for the EventDistributor interface. This interface, also very simple, has two methods – one to add a new consumer and another to remove a consumer from the list.

Event Filter Abstract Class

The code for the EventFilter class is shown in Listing 3. The class uses the EventListenerList class from swing (discussed above), but the rest of it is very straightforward. The constructor simply creates a list object to handle the consumers that will register with this object. The method consume is the abstract method that children will have to implement. The addEventConsumer method and the removeEventConsumer methods manage the contents of the EventListenerList object. Finally, a protected method called fireEventConsumers handles the actual sending of the event to all consumers currently registered in the list object. Derived classes will want to call this method if the event is deemed to be of interest to the consumer's children.

Event and EventContext Classes

Listing 4 contains the code for the Event class. The constructor builds the EventContext object and creates a hashtable object to handle the attributes to be added to this Event.

The methods getAttributeByName and setAttributeByName are used to assign attributes a value and obtain the value from them. They both operate on a string for the attribute name and an object for the attribute value.

The method isA takes a string that contains a context portion and returns a Boolean indicating whether this event is of that type.

The publish method of the Event class is left empty at this time. It will be implemented when we get to the distributed version of the application. For this implementation the test drivers simply call the publish method on the distributor.

The EventContext class is shown in Listing 5. Its constructor verifies that the URL passed in is of the proper type, and initializes its internal variables. The other two methods of importance are getComponent and nextComponent. The former returns a string with the current component of the URL. A component is defined as ending with either a forward slash (/) or a question mark (?) character. The forward slash is to separate static context information (e.g., Chart Update, FailedLogin). The question mark is used to separate dynamic information (much as it is in many CGI-based URLs). An example of this would be the identifier of the chart that has been updated.

Distributor

Listing 6 contains the BasicDistributor class (the real distributor will be provided as an EJB object in the next article but this one will work for local distribution). There are only two points to notice about this distributor. The first is that it is a subclass of EventFilter. The second is that its publish method simply calls fireEventConsumer.

Consumers

Listings 7 through 9 show three consumers. Two handle events of type "Security" (SecurityConsumer and FailedLoginConsumer); the third handles events of type "Chart Update." All three consumers print a message when they receive an Event whose context matches the one in which the consumer is interested. The FailedLoginConsumer performs a second check (the attack method) to see whether an oversimplified attack to break into the system is taking place. If so, it prints a message that an attack was detected.

Test Driver

Listing 10 contains a test driver to test these classes. It first creates an instance of the BasicDistributor class. Then all three of the consumers are registered, and finally some Events are created and published. The standard output generated by the test driver is shown in Listing 11. It should be run with and without the sleep call commented out to ascertain that the FailedLogin consumer works as described.

Conclusion

This code shows how simple it is to implement the basic functionality of this design. In the next segment I'll show how to extend this design on top of EJB so clients running in VMs distributed across the network can be both event consumers and event producers.

References

Matena, Vlada et al. (1998). Enterprise JavaBeans 1.0 Specification, Sun Microsystems, Inc., Palo Alto, CA, HYPERLINK http://java.sun.com/products/ejb/docs.html; http://java.sun.com/products/ejb/docs.html.

Gamma, Helm, Johnson and Vlissides (1995). Design Patterns. Addison Wesley. ISBN 0-201-6336-2.

CODE LISTING

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

About the Author

Brian Zimbelman is a snowboard instructor who moonlights as a senior architect at CyberPlus Corporation. He has been working on distributed systems since 1984 in C, C++ and now Java. Brian can be reached at bzimbelman@cyberplus.com.

bzimbelman@cyberplus.com

JMaskField

These advanced features make your user interfaces more user-friendly

by **Claude Duguay**

When you write user interfaces, you inevitably have to collect information from text fields and validate the data before you use it. There are several ways of handling validation. You can verify the text as the user exits the field by watching for lost focus events, or you can wait for the user to dismiss a window or dialog box by pressing a button, thereby validating all the fields at once. Both approaches are useful, but they can also lead to complex scenarios. Providing appropriate feedback and cursor positioning when invalid data is entered can often become complicated. Often, a better approach is to validate the information as it's being entered by the user – a technique known as keystroke validation.

This month's JMaskField goes much further than restricting the user to valid keystrokes by supporting the following advanced features:

- **Customizable Rules:** We use a regular expression-style syntax to define rules that determine whether a given character is acceptable for each position in the field. A mask can be defined with a mix of literal characters and validation rules.
- **Macro Characters:** We can make it possible to associate any character with an expression in order to support a more compact notation. For example, "#" might be assigned to an expression like "[0-9]" to represent numerical values.
- **Cursor Positioning:** The cursor is positioned intelligently after keystrokes, skipping literal characters that don't need to be typed in by the user.
- **Template Character:** You can define any character as a visual cue to indicate positions where the user hasn't yet typed or where a character was deleted. The default template character is the underscore ("_").

Figure 1 shows the classes we'll be developing and how they relate to each other. While this may seem like a lot, you can see by the diagram that most of the classes address tokenizing and parsing the mask. Once parsed, the validation is pretty straightforward and is handled by our extension to the PlainDocument class. JMaskField extends JTextField and merely adds some cursor movement code to make it more user-friendly.

JMaskField provides the high-level interface you'll use in your applications. In practice, you can provide a mask and template character in the JMaskField constructor and check the field output for template characters to determine if they were fully entered by the user at runtime.

Tokenizing Masks

To process the field mask we need to tokenize the text and build a parse tree from the token list. The parse tree elements are used to match the characters as they're being typed by the user. The Java class library includes two out-of-the-box tokenizers: StringTokenizer and StreamTokenizer. While these are useful, they just don't provide sufficient information when telling the user where the problems have occured. So we'll create a more flexible solution.

The MaskTokenizer produces a list of MaskToken elements. And, this MaskToken class, as seen in Listing 1, has two member variables: *pos*, an integer value that stores the offset from the beginning of the tokenized text, and *text*, a string value that stores the actual token. Because many tokens are single characters, we overload the equals method to handle both string and char inputs. This makes parsing easier when we need to determine what type of token we're dealing with.

Listing 2 shows the MaskTokenizer class, which has a single constructor that requires two arguments: an include string that identifies all delimiter characters which should be returned as tokens, and an exclude string that identifies all delimiter characters which shouldn't be returned as tokens. A math tokenizer, for example, might use an include string like "+-*/"and an exclude string like " ". This would return any nonspace sequence and consider the math operators separate tokens.

The MaskTokenizer provides hasMoreTokens and nextToken methods, just like the Java tokenizers, but nextToken returns a MaskToken instance. In addition, we provide an ignoreToken method to push back the current position after reading a token. This is useful with most parsers, which sometimes need to look ahead before determining whether the next token is relevant.

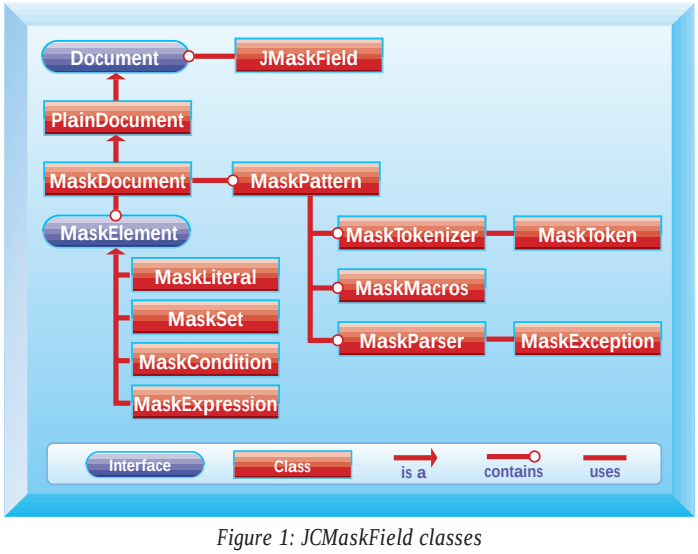


Figure 1: JCMaskField classes

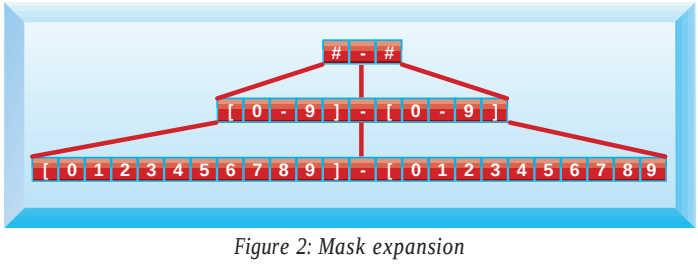


Figure 2: Mask expansion

Regular Expressions

Regular expressions are used heavily in languages like Perl and AWK, and are often applied by programmers either through command line searching with the GREP utility or in flexible search/find capabilities exposed in modern user interfaces. While these are often considered too complicated for average computer users, they fit quite nicely into the savvy programmer's bag of tricks.

Table 1 lists the characters we're interested in. Regular expressions have a couple of reserved characters that represent the beginning and the end of a line, as well as sequence modifiers. These aren't directly applicable to our solution, so they're not included in this table. If you're familiar with regular expressions, you'll notice that we've changed the character for NOT operators. This is nothing more than artistic license. Naturally, if you prefer something else in your application, you're free to change it to whatever you like.

We'll take a look at the syntax in a moment. In the JMaskField widget, a mask is provided in the form of a string. The string can be a mix of literal characters – which may not be edited – and rules expressed using our regular expressions subset. To distinguish between rules and literals, we delimit rules with curly braces. Table 2 shows a few valid masks with a brief explanation for each.

The last example shows how macros can be used to make this syntax more compact. Let's take a quick look at the parser.

Parsing the Rules

Once the text has been tokenized, we can organize it by constructing a parse tree. When errors are encountered, we throw a MaskException (as shown in Listing 3) to tell the caller what was expected and the text position where the error occurred. This makes it a lot easier to deal with syntax errors before they become problems. The text offset position is especially informative and makes addressing any occurring problems much easier.

The rule parser uses several supporting classes to represent the resulting item list. Each of these implements the MaskElement interface from Listing 4, which enforces the use of a toString and a match method. The toString method is useful for debugging, so we can see the structure by just writing it out. The match method tests a character for validity and will get used in the document class we're implementing later.

Here's a quick look at the syntax using the BNF format:

```
<element> ::= '{' <condition> '}' |
  <literal>
<condition> ::= <expression> [ <conjunction>
<condition> ]      <conjunction> ::= '&' |
  '|'
 <expression> ::= '(' <expression> ')' |
  <character-set>
<character-set> ::= '[' [ '!' ]
  <characters-list> ']'
```

BNF allows us to represent the syntax in a manner which is very close to the way we need to program the parser. The productions above can be easily described in English. Each production involves an element on the left and options on the right. In BNF the options are separated by a "|" character and may include optional elements that are delimited by square brackets. Thus the first production means an element is either a condition (delimited by curly braces) or a literal.

The second production means a condition is an expression, followed by an optional conjunction and another condition. The conjunctions are either the *or* ("|") character or the *and* ("&") character. The expression production is there primarily to allow parenthesis-delimited nesting – exactly the way mathematical expressions can be given precedence by wrapping them in parentheses. If no parenthesis is present, we expect a character set.

We consider a character set to be a special case in our parser by expecting a set to be delimited by square brackets and to be optionally negated, using the "!" modifier. Character ranges aren't handled as tokenized elements. It's easier to consider anything tokenized as a single character sequence and to process sets with the parser. When we run into a set, we traverse the characters and handle dash ("–") delimited character pairs by dynamically expanding them so that the resulting set is explicit. This makes later matching more efficient.

Listing 5 shows the MaskLiteral class, which represents literals and stores the character internally. The match method simply does a direct comparison with the test character. Macro characters are con-

Description	
()	Precedence delimiters
[]	Character set delimiters
-	Set range delimiter
!	NOT prefix operator
&	AND operator
|	OR operator
Any other character	
Character literal	

Table 1: Reserved regular expression characters

PRIVATE Mask	Description
"{[A-Z]}{[a-z]}{[a-z]}{[a-z]}"	One uppercase and three lowercase letters
"{[!0-9]} {[!0-9]} {[!0-9]}"	Any three nonnumerical characters, separated by spaces
"{[02468]}{[13579]}"	Any even number followed by an odd number
"(###) ###-####"	A phone number when macro '#' is defined as "[0-9]"

Table 2: Valid mask examples

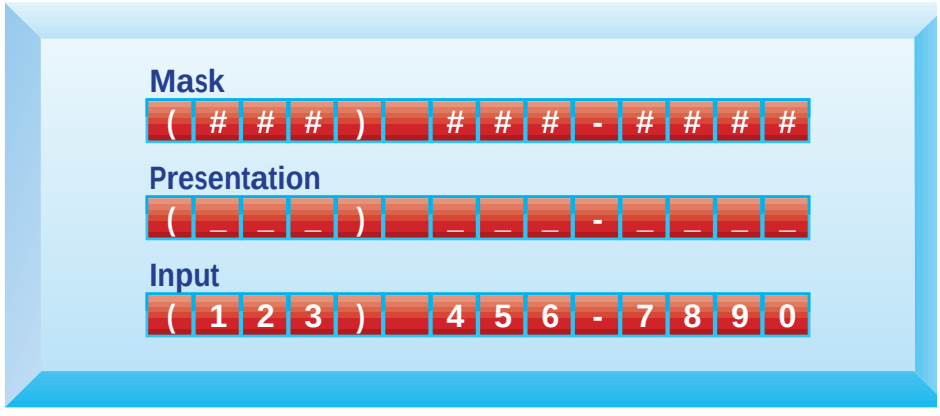


Figure 3: Mask presentation

sidered literals until they're interpreted at runtime. This allows us to change the macro definitions without requiring the mask to be parsed again, thereby increasing our flexibility.

The MaskSet class is shown in Listing 6. A set of characters is made explicit by the parser, expanding hyphenated ranges into a string list. The parser automatically handles inverted ranges if the value of the rightmost character is less than the value of the leftmost character. The only additional information required in our MaskSet representation is the negation marker if a NOT operator was used and stored as a Boolean value.

Listing 7 shows the MaskExpression class. Expressions are just wrappers designed to handle precedence. The match method calls the encapsulated MaskElement at comparison time. MaskCondition is more interesting. Listing 8 shows how it stores a Boolean value to indicate whether an AND or an OR conjunction is used. We keep a couple of constants around to make the parser code more readable. The match method uses the Java logical *and* ("&") and *or* ("|") operators to resolve the function. The left and right arguments are resolved by calling their own match methods.

The MaskParser, shown in Listing 9, is implemented as a separate class so it can be used by both the MaskMacros and the MaskDocument classes. There isn't enough room here to say much about recursive descent parsers, other than the fact that they operate much as the name implies. They use recursion to build a tree structure and descend to parse any nested structures. As mentioned earlier, the structure of the methods in MaskParser closely resembles the structure of the BNF notation used to represent the syntax.

Extending the Model

One of the objectives I had when I decided to implement the JMaskField control was to make it both powerful and easy to use. This is one of those standard programming dichotomies that's difficult to resolve and requires some thought. The best option I was able to identify was to make the syntax for defining rules ultimately flexible. That makes it powerful, providing a mechanism for abstracting those rules in simple form. This mechanism is implemented in the form of character macros.

Figure 2 shows how a simple mask gets expanded to a parsed expression, and finally to explicit character sets.

Listing 10 shows how the MaskMacro class is really little more than a hashtable that stores an association between a given character and a MaskElement representing the rule(s) to be applied. When the MaskDocument in Listing 11 runs into a literal character, it checks to see if there's a rule associated with it in the MaskMacro model. While it's always possible to define masks using the curly brace syntax, you can see that it's much easier to define your own rules and assign them to macro characters.

The Document Model

The MaskDocument class extends the PlainDocument class in the JFC and can be assigned to any JTextComponent. The JMaskField class is presented in Listing 12 and extends JTextField, implementing additional behavior to handle intelligent cursor movement. Let's take a quick look at the MaskDocument class before we cover the JMaskField code.

To provide visual feedback, we generate a template for the mask expression. To keep things simple for the user, we'll use an underscore as a placeholder for nonliteral characters. The underscore is the default template character, but you can easily change it if you prefer something else. Figure 3 shows how the mask presentation and user input parallel each other.

The MaskDocument class implements supporting methods to handle the template and to make character matching easier, but it primarily implements the remove and insertString methods required by the JFC Document interface. The Document interface has two methods we have to override in order to get the behavior we need.

The insertString method is called any time new data is entered in the text field, typically after every keystroke. The remove method is called whenever text is deleted.

The insertString and remove methods can pass several characters at the same time, as would be the case in cut or paste operations. To support these, we break each string into single characters and handle them recursively. The net effect is that a parse operation may not complete if some of the characters don't match, but the field will still handle as many characters as possible. Each character, processed individually, is tested by the match method and allowed to replace template characters in the insertString method. A character is replaced by a template character when being deleted in the remove method.

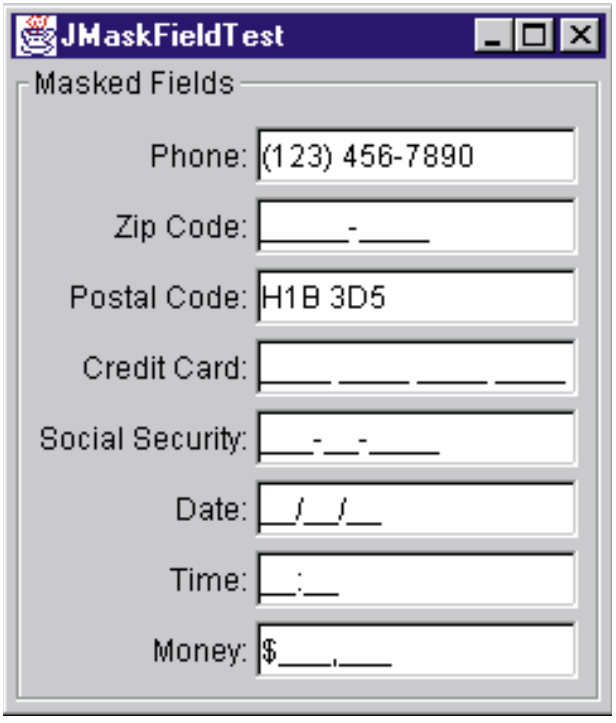


Figure 4: JMaskField at work

Summary

Figure 4 shows JMaskField in action. The Phone and Postal Code fields have already been entered and the other fields demonstrate a mix of literal and template characters that provide visual cues to guide the user through a successful experience. When inappropriate characters are typed in, the user hears a beep and the character is rejected.

JMaskField provides a flexible mechanism for constraining character entries in a text field. Because it uses standard Java strings, any valid character pattern can be applied to define the data mask. The simple, regular, expressionlike syntax lets you define arbitrary character rules. Extending this model to support character macros adds even more flexibility and the template view provides useful feedback for the user. Together, these elements provide you with yet another tool to make the user experience as pleasant as possible. Use it in good health.

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 he currently leads the development team at Atrieva Corp. You can contact him with questions and comments at claude@atrieva.com.

claude@atrieva.com

If It Ain't Broke – Don't Fix It

A look at builders, debuggers – and the nature of inquisitiveness

by **Alan Williamson**

HALT! Just stop right there! You've probably stumbled across this column while merrily thumbing through this magazine, and you're now wondering what this lump of words is all about. You may have noticed this column in previous issues but couldn't be bothered to read it. After all, who'd blame you, since there aren't any pictures or fancy diagrams to support it. It's got to be dull, right?

But why not be daring? Go on, lend me 10 minutes of your time and let's see if I can entertain you as I give you some Java enlightenment. If you don't like it, then hey, I won't hold it against you. We won't even speak of it again. Deal?

Well, last month saw me in Tokyo. Again. I was there giving a series of talks on the merits of using Java at the server side. The talks were geared around Java Servlets and the benefits they can offer over and above alternative technologies such as CGI or Microsoft ASP. The presentations highlighted and talked listeners through many of the large-scale case studies that we (here at N-ARY Limited) have installed. At the end of each talk there was an in-depth question-and-answer section. And that's where the fun began.

But before I continue on to the core of this month's column, it's time for that characteristic-parallel thing to occur. This is where I take a personality trait and apply it to our consciousness of Java. This month I'm going to go for inquisitiveness – the trait of continually asking questions.

Inquisitiveness is something we're familiar with when we're young: "Mum, why is the sky blue?" and "Mum, where do babies come from?" As soon as we develop the ability to speak, we start learning. Learning is achieved by looking and pointing at things and continually asking what must seem like silly questions to our parents. It's a shame, but most of us lose this quality somewhere in puberty.

Back to Japan

So after a two-hour presentation on the merits of Java Servlets, I was all geared up for a whole host of questions on the Servlet API. Maybe some questions on how to handle special HTTP requests or about different browsers. Maybe even a question on how the Servlet API is used within application servers. But what I thought and what I was asked were two different things. I think, in all honesty, the percentage of questions asked on the Servlet API was around the 5% mark. The remaining 95% was on basic Java principles. I thought: How wonderful.

As discussed in this very column a number of issues ago (get the back catalog – a sound investment!), the software industry in Japan isn't as far along as its consumer electronics. They're still very much at the early adopter's stage of development – which is actually good. They haven't suffered the same side effects of continually moving JDK versions as we have, since the majority of them started on 1.1.

So there I was, promoting the whole Java Servlet thing, hoping to persuade people to not go down the CGI route of development, and what they were asking about was fundamental Java. I met a lot of developers in Japan and nearly all of them asked me the same question, "What do you use to debug servlets?" This line of questioning surprised me. At first I thought it was just the one time, but when the same question was posed a second and then a third time, I thought: Okay, maybe we've missed something in our teachings.

The answer I gave them was simple and took many of them back a bit as they thought I was joking: use System.out.println(...). This is the most sophisticated debugging tool I've ever used. It holds many advantages over many other techniques, in that it's completely portable and it doesn't matter which virtual machine you're running on – or which platform. It will work. It's simple – nothing too complicated and you don't need to worry about exceptions. In a forever increasing world of complexity, the ability for simplicity to shine through still exists.

Back in my old days of coding C and C++, and after my university years, I had the pleasure of working with a mentor who taught me a thing or six about real developing. The months I spent with him changed my whole outlook on development forever. I thought: Why couldn't they have taught this in university? Dave Forth was his name, and he believed that debugging tools were only there for lazy developers. Now at the time I was a big fan of Turbo C and its inline debugger. The ability to trace through execution steps was a godsend for me. So, to have this man turn my world upside down by dismissing debugging tools, I thought: Oh no, I'm lost now!

But you know, the more I thought about it, the more he was right. For example, you don't see house builders using debugging tools. The house either stays up or comes down after the first storm. After a house has been erected, the master builder doesn't clear the building site and start up a house debugger. (What a house debugger might actually look like staggers the imagination.) So how can the builder be sure the house won't fall down?

Well, one of the new waves that's gripped the developing community in recent years is that of object orientation. We've all experienced objects – you can't develop Java without being aware of them. One of the goals of an OOD (object-oriented design) system was to reduce the complexity of a system and thus make it less prone to errors. A good OOD system shouldn't require that much debugging. After all, you're developing small units that can be individually stress-tested. But it's not a new way of operating. The computing industry has just taken an old idea, given it a fancy name and heralded it as the new and improved way software should be coded.

Industries have been practicing OOD for hundreds of years. Let's look at our builder again. It's a perfect example of OOD in the real world. The builder doesn't build each individual brick. Nor does he test the strength of each brick. He's already bought into the technology (yes, even a brick has technology, i.e., the technology to engineer it to the standards that are suitable for building). Now he places these bricks together in the manner in which the bricks' API has determined for how it's going to interact with other bricks. Anyone who's played with Legos or any other small building blocks when young knows that to build a strong wall, you don't place the blocks one on top of each other. Instead, you interlace them. This is a brick API (or definition) of how you can best use a brick.

The builder uses many such materials, and it's the skill and knowledge of the builder that determines how bricks and other materials go best with one another. But he knows that once the brick is put in its place, he doesn't need to worry about it again, i.e., he won't

need to debug that brick at a later date. Granted, there are builders who aren't particularly skilled at this, and I'm sure some sort of house debugger would be a godsend to them.

But what if something does go wrong with the house? What can the builder do? Well, an experienced builder would know what to look for in order to fix the problem. He'd spot the telltale signs. Well, there's no reason why a developer can't operate on the same principle.

But it's relatively easy to find problems in a house. You can walk around it, get inside it and discover the problems. What can a developer do? In order for the developer to be able to fix a problem, he must first try to find it. But many fixes only solve the symptoms – not the problem. For a developer to see what's going wrong, he must see how sections react, how variables are changed.

And you can do this with a debugger....

Yes, you can – there's no denying that. But as the physicist Leo Szilard questioned in 1929: "Can you know all about the world without changing it?" He was talking about thermodynamics at the time, but the same thing applies to our world. It's like any measuring tool – it's never completely impartial. The very presence of the measuring tool creates a disruption in the natural flow. For example, a small turbine placed in a stream to measure flow rate will offer a small amount of resistance to the stream. Granted, this small resistance may never be detected, but just because it can't be measured doesn't mean it's not present. Software debuggers are the same way.

Have you ever tried to solve a memory leakage problem or a threading problem with a debugger? Ever wondered why you could never reproduce the problem within the debugger, but as soon as you ran it on its own things suddenly started crashing? Or even worse, you're developing as you go along within the debugging environment, only to experience weird and wonderful crashes when it's run stand-alone?

This is because the very presence of the debugger has introduced a new unknown element to your environment. An unknown constant that you've no control over is infecting your world. It sounds a bit drastic and melodramatic, and it won't be the case for every single bug you're trying to track, but why take the risk?

Back to our builder. Once he's spotted the error, fixing it is generally not a major deal, unless, of course, he's spotted a fundamental flaw in the overall design of the system. For example, maybe using paper-based bricks for the foundation wasn't that great an idea after all! The developer needs to gain the same insight.

The simplest way for the developer to discover problem areas is to try to insert one of the least intrusive tools possible: System.out.println(...). For example, Java has given every object the ability to print to string through the java.lang.toString() method. But how many of you actually override this simple method in your classes? I'd guess very few. To those of you who don't, why don't you? You can place a whole host of useful information about the classes state in here, so should the day come that you need to print out some information, you can simply make a call to your toString() method without worrying about littering your code full of System.out.println(...) statements.

It's this simplistic view of looking at things that generally yields the quickest results. You have to be inquisitive with your code. Learn to ask it silly questions and who knows, you may be surprised at what answers come back. As the textbooks say, it's about 95% design and 5% coding (...and 70% debugging!). By using the inherent tools Java has provided, we can hopefully remove this unwanted 70% – or at least reduce it significantly.

Book Review

Since this is the beginning of a new six months, I thought I'd introduce a new section to the bottom of each column. Over the last six months we've addressed many different issues ranging from development to running a complete Java business. I'm a single voice in a sea of noise, so I'm going to recommend a book to you each month. Each month the book will have relevance to a column that was previously in my "Straight Talking" series.

To start this mini book review off, I'm going to recommend a book I purchased on University Drive in Palo Alto while in California earlier this year. This book complements my past column on start-ups, (***JDJ*** Vol. 3, Issue 10). *Losing My Virginity* by Richard Branson is an autobiography showing the rise of Britain's top businessman and his empire, Virgin. It's a fascinating read and very inspirational to anyone thinking of starting a new business venture. Branson is extremely honest, admitting to his mistakes and how he'd probably do things differently if he had the opportunity.

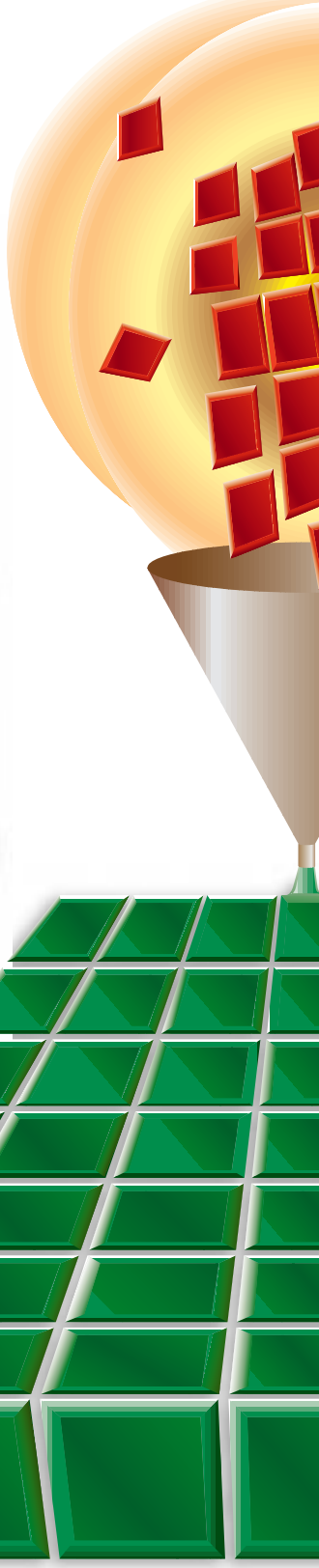
It was interesting to read how a noncomputing empire was built with the same ethic that many developers hold dear to their hearts – business should be fun, and not all shirts and ties.

About the Author

Alan Williamson is CEO of N-ARY Limited, a UK-based Java software company specializing solely in JDBC and Servlets. He recently completed his second book, which focuses on Java Servlets. Alan can be reached at alan@n-ary.com (www.n-ary.com).

alan@n-ary.com

Design Patterns in a Java Interpreter



How they can aid in the design of complex software

by **Gene Callahan and Brian Clark**

This article describes our use of design patterns to create an interpreter in Java, and shows how it can be built in a "pure," object-oriented fashion. The patterns we use are from *Design Patterns: Elements of Reusable Object-Oriented Software* by Gamma, Helm, Johnson and Vlissides, published by Addison Wesley in 1995. (We'll refer to this book henceforth as *DP*.)

What Did We Build?

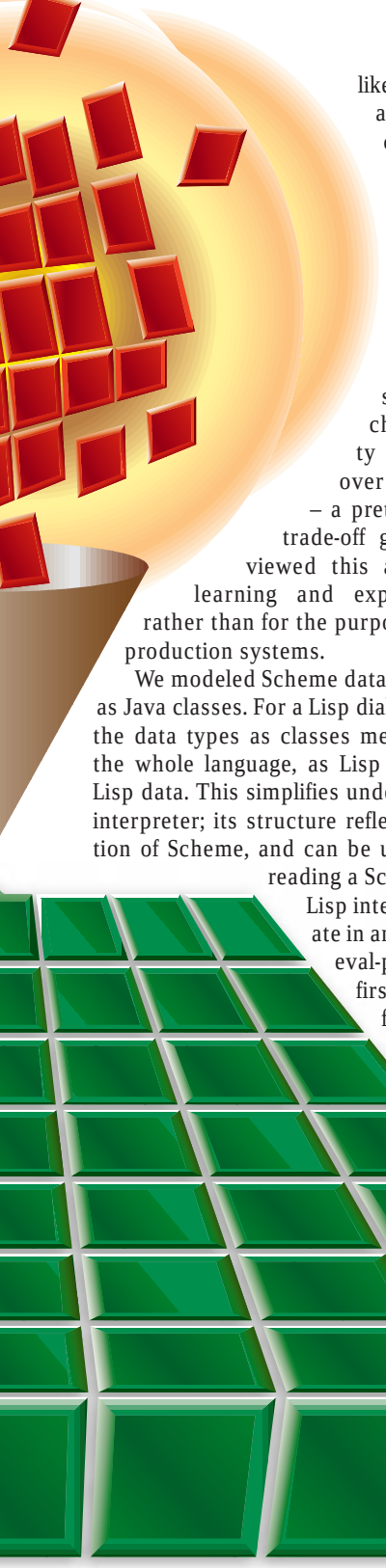
HotScheme is an online, multiplatform interpreter of the Lisp dialect Scheme, with GUI front-end and interactive Internet capabilities. It's currently implemented as a Java applet, although the core of the interpreter is independent of the front end. At present it's only a partial implementation of ANSI Scheme, intended for use as an educational tool for learning Scheme, GUI design and interpreter implementation.

The project came into existence while we were working with Dr. Jo Anne Parikh of the Southern Connecticut State University Computer Science Department. The interpreter was originally written in C++ and later ported to Java. We found that the port was not difficult, and that for our purposes Java had several key advantages over C++. It made garbage collection a breeze, exception handling more elegant and allowed us to easily add features to the language, such as fetching the contents of a URL and creating objects from the Java AWT. And of course, its being able to run as an applet from a Web browser handled the problem of distribution. We plan to take further advantage of Java's built-in networking capabilities and make more AWT features available from Scheme.

The Patterns

Interpreter: Modeling Scheme Grammar Directly in Java Objects

In the Interpreter pattern a class represents each grammar rule in the language. The interpreter's parse tree, which is its structural representation of the program under execution, is built as a Composite (see next section) of simpler grammatical elements. We extended the Interpreter pattern to include interpreter input, borrowing an idea from Bjarne Stroustrup's *The C++ Programming Language* (Addison Wesley 1993). Instead of the core logic being in a large state machine, as in a procedural interpreter, it is dispersed through a hierarchy of classes. Objects in HotScheme know how to construct themselves from an input stream, assemble themselves into a parse tree and return the value they represent. There is a trade-off here: a more traditional interpreter,



like those generated by the combination of lex and yacc, will run faster, but they may be a little harder to understand. It was for that reason that we chose simplicity and clarity over performance – a pretty reasonable trade-off given that we viewed this as a tool for learning and experimentation, rather than for the purpose of creating production systems.

We modeled Scheme data types directly as Java classes. For a Lisp dialect, capturing the data types as classes means capturing the whole language, as Lisp programs are Lisp data. This simplifies understanding the interpreter; its structure reflects the definition of Scheme, and can be understood by reading a Scheme manual.

Lisp interpreters operate in an endless read-eval-print loop by first getting input from the user, evaluating it and then returning the result of the evaluation. We discuss the read phase of this loop in the Abstract Factory section below. Evaluating the parse tree is simply a matter of calling Eval() on the SchemeObject returned by the read phase. Leaf nodes in the tree (Scheme atoms, such as numbers, strings and symbols) return themselves as their value or, in the case of symbols, return the object they label. Calling Eval() on high-level nodes will trigger a recursive evaluation of all nodes lower in the tree, yielding a single return value – of a type descended from SchemeObject – for that portion of the tree. Similarly, this result is printed by calling Print() on the object returned by Eval(). If this object is a composite, it will call Print() on its components. This makes the top-level code so simple that the body of this loop, the heart of the interpreter, is a single statement:

```
// term is the current terminal, global_env
the lexical environment:
term.println(((SchemeObject.make(term,
SchemeObject.START,
    global_env)).Eval(global_env)).Print());
```

The meat of the evaluation phase is in the List.Eval()method (see Listing 1). This is because the most fundamental Scheme activity is function application. When a list is evaluated, its default behavior is to treat the first item in the list (the car of the list) as a function to be applied to the rest of the list (the cdr) – the arguments to the function. As Java lacks the varargs feature present in C and C++, having a list data structure directly available in Java greatly simplified writing Java functions that handle the variable-length argument lists required by Scheme. Every Java method that implements a Scheme function takes a list of SchemeObject elements as its single argument, and pulls apart its "actual" arguments itself.

It's important to note that List.Eval() calls Evargs() (evaluate arguments) on the rest of the list before passing these arguments to the function. (We use an auxiliary function, Evargs(), rather than Eval() itself, because a second call to Eval() would treat the first member of the rest of the list as a function – not at all what we want!) For this reason certain Scheme forms, called *syntax forms*, can't be handled by an ordinary function application. Consider the case of an *if* statement. The ANSI Scheme specification for if says that it evaluates its first argument and, if true, returns its second argument; otherwise it returns its third. But the specification also states that whichever branch is not returned must not be evaluated. If we used an ordinary function application to evaluate if, it would be too late! We'd have evaluated the arguments before even calling if. Thus these syntax forms are handled by special cases in List.Eval(). The objects created to represent these forms in the parse tree are all descendants of the HotScheme class SyntacticalForm, in keeping with our use of classes to capture grammar.

Composite: Building the Parse Tree

The intent of the Composite pattern is to allow clients to treat individual objects and composites uniformly. The parse tree that HotScheme builds to represent the command it is interpreting is an instance of this pattern.

Lisp programs are built from Lisp's main compositional structure: the list. The fact that a Lisp program processes lists and is also made up of lists lends an elegant simplicity to a well-built Lisp interpreter. Lists are a basic data type, and can generally appear in the same places as atoms such as integers and strings. This allows us to use a Composite to represent the parse tree. We represent lists as SchemeObjects that hold references to other SchemeObjects, which can themselves be lists. A client calling an object's Eval() or Print() method need not worry about whether it is dealing with an atom such as an integer or a composite such as a list – the call is made the same way by the client, with the composite recursively passing the call on to its components, if necessary. For an example of how Composite simplifies handling aggregate entities, see the implementation of List.Print() in Listing 2.

Abstract Factory: A Common Ancestor Creates All Scheme Data Types

As mentioned above, objects know how to read themselves from a tokenized input stream and assemble themselves into a parse tree. The class SchemeObject, in its role as an Abstract Factory, is the place where this knowledge resides.

The pattern *DP* refers to as *Abstract Factory* is also described in James Coplien's *Advanced C++* (Addison Wesley 1992), where it is called the *Exemplar idiom*. An Abstract Factory allows clients to create subclasses of a class without specifying which subclass to create. To achieve this, SchemeObject itself determines which Scheme data type we are reading. When its static make() pseudo-constructor passes a reference to a terminal for input and to an environment for interpretation, it will return a (properly subclassed) reference to whatever object type it finds waiting for it on the input stream (see Listing 3). SchemeObject asks the lexical analyzer for the next token. It looks at the type of token it receives to see which of its subclasses to instantiate. This can be done recursively so that when we find a composite object like a list on the stream, the object returned will have constructed the elements of the composite and will be holding references to them.

Abstract Factory posits that clients will deal only with the abstract interface provided by the factory class, and not call subclass-specific methods. The SchemeObject is this abstract interface in HotScheme, and is the base class for all concrete Scheme data objects. This is important, because

many Scheme functions, such as predicates like list? and number?, can operate on any type of Scheme object. Also, Scheme lists and vectors are heterogeneous collections, and require a common base type to hold references to. Because of this level of abstraction, clients don't need to know about new Scheme data types as we add them to the system.

Having SchemeObject as an abstract base class also helped when it came to error handling. Because Scheme is not strongly typed, any function might pass any data type for any of its arguments. However, this doesn't mean that every function can handle any data type! Many combinations should produce a runtime error. For example, the Scheme command first makes sense only when its first argument is a list. It's meaningless to ask for the "first element" of an integer. HotScheme handles this by throwing exceptions in the base SchemeObject class for most methods. A call to SchemeObject.first(), by default, throws an exception that states that the object the method was called on is not a list. We overrode that method only in the List class. This eliminates the need to scatter type-checking code throughout the system – in this case a big gain in both simplicity and performance.

Command: Scheme Built-in Commands as "Functors"

Often, commands in an interpreter are stored in a jump table that associates function names in the source code with function addresses, or *jump points*. When a symbol matches a name in the table, the interpreter "jumps" to that address to execute the function there. However, when we went to implement the built-in functions, we found that there was no straightforward way to create a jump table in Java. This is because there are no global functions, and no way to get a pointer to a member function. We wound up wrapping each function in what James Coplien terms a *functor* – an example of the Command pattern from *DP*. Each built-in command has its own class. To execute the command, we call the class' Apply() method, passing it its arguments – wrapped in a Scheme list – and an environment in which interpretation will take place. (Conveniently, this is also how we execute a user-defined function.) We instantiate one object for each built-in command when we initialize the interpreter, and store this instance in the symbol table as the value associated with the symbolic name of the command. Thus the object representing the function is first stored in a hash table associated with the key "first." The code (first '(a b c)) will cause the interpreter to look up that object and call its Apply() method, passing it the list (a b c). The Apply() method returns a SchemeObject, in this case a. At the point of execution the interpreter neither knows nor cares whether it is executing a built-in or user-defined function – that knowledge is stored in the function object itself. The drawback of our solution is that it has led to a large number of classes for built-in commands. See Listing 4 for an example of one of these functors.

Façade: Our Terminal Interface

The Façade pattern hides a number of complex interfaces behind a simpler, higher-level interface. We employed it to hide I/O details within the class LispTerminal. Our terminal interface is minimal, with little coupling between the UI and the interpreter. In addition to our GUI version we've implemented a version for a Java character terminal, and it would be trivial to make a version that, for instance, interpreted code coming in on a socket. Instead of having the interpreter attempt to deal directly with character terminal, AWT, Swing, socket, Accessibility and other interfaces, the LispTerminal class presents a few abstract operations – like reading and printing – that the interpreter needs. The interpreter is passed an object that is a descendant of LispTerminal, to which it will direct I/O requests. Thus it is the creator of

the interpreter instance, not the interpreter itself, that decides how I/O will be performed. For our character terminal version, input is read straight from the terminal. GUILispTerminal buffers keyboard input from HotScheme's input field and sends it all to the interpreter once the "Evaluate" button is clicked. To perform output, the interpreter calls the Print() method of the terminal passed to it, and the different terminal types output the text correctly. LispTerminal also provides a pushback buffer when the tokenizer has to read "too far" to tell when it has completely captured a token. (For example, a tokenizer can't tell that it's done reading a number until it reads the first character that is not a digit – one too many! The tokenizer needs to "push back" this extra character onto the input stream so that the next call for a token will read it.) Since lower-level I/O objects may not provide this capability, the Façade abstraction again simplifies our interfaces. See Listing 5 for the definition of LispTerminal.

How Did Patterns Help?

Employing patterns lent shape and coherence to our high-level design. Without being able to think about the interpreter using these patterns, the complexity of its design would have expanded beyond our grasp. The patterns gave us a way to think of the interpreter as a number of very high-level constructs, the details of which we could ignore when considering the interpreter as a whole.

The use of patterns is also crucial in communicating the design. In our discussions it was immensely helpful to be able to give names to the ideas shaping our work. To say "We'll employ an Abstract Factory to create Scheme types" captured a large piece of design in a simple, succinct statement.

Finally, by densely combining patterns in a small design space, we began to glimpse the poetic quality that Christopher Alexander, in *A Pattern Language* (Oxford University Press 1977), asserts this "compression" of patterns can produce. As we wove these patterns into our design, the program began to surprise even its authors in the way new features effortlessly emerged from the structures we had already created. And this sense of adventure and elegance is what can make our profession a fulfilling one to pursue.

We welcome communication from anyone interested in contributing to this project, and from any computer science departments or other educators who would like to deploy HotScheme at their institution.

Resources – URLs

Patterns Home Page: HYPERLINK http://hillside.net/patterns/
http://hillside.net/patterns/
Pattern FAQ Page: HYPERLINK http://gee.cs.oswego.edu/dl/pd-FAQ/pd-FAQ.html
http://gee.cs.oswego.edu/dl/pd-FAQ/pd-FAQ.html
Christopher Alexander: An Introduction for Object-Oriented Designers: HYPERLINK http://gee.cs.oswego.edu/dl/ca/ca/ca.html
http://gee.cs.oswego.edu/dl/ca/ca/ca.html
HotScheme: www.stgtech.com/HotScheme

CODE LISTING

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

About the Authors

Gene Callahan is president of St. George Technologies, where he designs Internet projects. He has written for ***Computer Language, Software Development*** *and* ***Web Techniques****, among others. He can be reached at gcallah@erols.com.*

Brian Clark is a software engineer residing in Virginia. His current focus is on the application of design patterns on UI and middle-tier design using Java. Brian can be reached at bclark@crosslink.net.

gcallah@erols.com bclark@crosslink.net

JAVA AWARD WINNING JAVA PRODUCTS

JAVA DEVELOPER'S JOURNAL JDJ EDITOR'S CHOICE AWARD

Every year, ***Java Developer's Journal*** presents two types of awards – the ***JDJ*** Editor's Choice Awards and the ***JDJ*** Readers' Choice Awards. These awards are designed to honor and recognize the leaders in the Java world – specifically, those companies and products that the editor and readers feel are the best of breed.

The Readers' Choice Awards are based on nominations submitted to our Web site and by e-mail. There are a large number of categories for which a product can be nominated, including Best Bean, Best Application Server, Best Development Environment and a host of others (see below). The Readers' Choice Awards are presented to the products that receive the most votes in a particular category, with ties being decided by the editor, who reserves the right to select multiple winners if the situation warrants it.

The Editor's Choice Awards are selected by the editor-in-chief of ***JDJ*** – which is me, Sean Rhody. For each category in question, I did research to identify and compare the products that were relevant. For example, for Application Servers I looked at the servers that were shipping at the time of selection and that support Java as a primary component development language. I compared features such as model support (COM, EJB and CORBA), Load Balancing and Clustering.

For the category Best Development Environment, I looked at things like support for two-way coding, ease of use of the interface and support for team development. In each case I tried to identify the product that was the best overall in my comparisons. In some cases I didn't have a clear winner, or I didn't know enough about the competition in the category to make a selection. In those instances I left the selection of a winner to the readers.

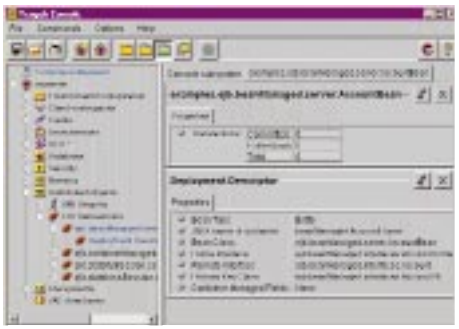
Congratulations to all of this year's winners!

☆BEA WebXpress
☆Category: *Application Server*
☆Winner: BEA WebLogic

BEA WebLogic is a Java application server for developing, integrating, deploying and managing large-scale distributed Web, network and database applications. With its comprehensive support for Enterprise Java Standards, WebLogic protects user investment and makes it possible to build portable, scalable applications that interoperate seamlessly with other applications and systems.

The BEA WebLogic application server offers the critical front-end Web component of BEA Systems' end-to-end enterprise middleware solution. BEA WebLogic:

- Fully implements 10 of the 12 Enterprise Java APIs, including JDBC, EJB, RMI, event management and JNDI
- Provides a comprehensive implementation of the Enterprise JavaBeans 1.0 specification
- Provides tools to aid in the creation and management of Enterprise JavaBeans, permitting the hosting of both custom and off-the-shelf business components
- Provides support for persistency to multiple databases
- Works easily with industry-leading databases, as well as Microsoft Visual Basic, Visual C++, Active Server Pages and COM
- Works easily with industry-leading development tools, including Visual Café, JBuilder, Supercede, J++ and Visual Age
- Deploys and manages applications to ensure scalability, availability and security



BEA WebLogic extends leading Java IDEs to support the development and debugging of multitier Java applications. WebLogic's implementation of Enterprise JavaBeans technology makes it easy to encapsulate business logic as secure, transactional components.

The BEA WebLogic application server is an extensible framework that allows any standard Java application to be "snapped in," including the Java client bindings provided for most legacy systems. BEA WebLogic's multitier JDBC implementation allows a Java application to access and update databases from anywhere on the network. BEA WebLogic also allows any Microsoft COM object to be easily plugged into the WebLogic framework. BEA WebLogic provides support for CORBA IIOP, IDL-compatible services and bidirectional interoperation.

Java applications hosted by BEA WebLogic can be replicated in a cluster with no additional programming. For scalability, WebLogic balances the load across available instances of the replicated service. For fault tolerance, WebLogic also replicates state information so that an outage can be completely masked from both users and applications. Network security is ensured through optional encryption, authentication and authorization based on the RSA Secured Sockets Layer, X.509 certificates and access control lists.

BEA WebLogic provides centralized management for large distributed configurations of clients and servers through a single comprehensive view of the overall system. Its pure-Java graphical management console allows remote monitoring, integrated logging and dynamic application partitioning. Its Zero Administration Client supports automatic distribution of software applications and Java applets.

☆Cloudscape
☆Category: *Database*
☆Winner: Cloudscape Database

Cloudscape offers the first zero administration database that's optimized for embedding in applications to be deployed outside the corporate "four walls." Cloudscape Database is a full object-relational DBMS, providing SQL-92, plus the ability to extend the system easily with Java class libraries to support complex data and logic. To support

the needs of next-generation Java applications, Cloudscape Database is designed to be pervasive, deployable, manageable and extensible.

Written in 100% Pure Java, Cloudscape Database can be embedded in applications at any level in the architecture, allowing data management to pervade the entire system, wherever needed, from server-class machines to laptops. Cloudscape is also positioned to support the emerging lightweight device market by providing database support for platforms such as Windows CE. Optimized for the new mobile and distributed applications, Cloudscape makes no assumption that database management skills are available at end-user sites, whether the end user is an employee, partner or customer.

With an extremely small footprint, Cloudscape is agile enough for deployment over the network. It is designed to meet the needs of companies that see the intranet and extranet as the most cost-effective mechanisms for delivering core business applications and products. Remote and embedded applications are managed from a central point. Supporting the database is Cloudscope, a tool for the database developer.

Cloudscape recently released the beta version of Cloudscape Application Synchronization, advanced technology for replication offerings that guarantees application-level consistency from existing corporate applications to the occasionally connected individual user or workgroup. A key to its success is its low cost of deployment and maintenance of distributed applications using the Cloudscape architecture.

☆DTAI

☆Category: *Framework*

☆Winner: LEIF

DTAI's lightweight extensible information framework – LEIF – leverages the best and most advanced features of Java to deliver a complete platform for information integration and data visualization. LEIF combines advanced JavaBean technology (InfoBus, BeanContext and JavaBeans Activation Framework) with intelligent data awareness through Java's reflection mechanisms. The LEIF architecture provides the means to gather information simultaneously from a variety of remote sources, using any protocol (CORBA, RMI, JDBC, etc.), to translate that information into a common object representation and then process and display that data intelligently using independently developed graphs, maps, spreadsheets and other visualization tools.

As a client, LEIF connects to servers via independently developed extensions called LEIF Producers. The LEIF Producer APIs allow LEIF to dynamically discover the objects, attributes and functionality of new data sources. LEIF Producers are responsible for all external communication and storage responsibilities. Even local file access is handled through a File Producer. As a result, LEIF does not mandate any particular data storage model (e.g., file-based, object or relational databases) or distributed computing architecture (e.g., RMI, CORBA or COM). If a producer is developed to support a particular storage or communications implementation, then LEIF supports it as well.

The LEIF framework provides a central display model that serves as a runtime reference to all available data. The display model pulls data from the LEIF InfoBus and builds a hierarchy – or tree structure – of all data supplied by a LEIF Producer. The display model also provides a flexible display filter capability. As data enters the display model, it's tagged with display attributes (e.g., color, symbol, line style) based on the filter criteria specified by the operator.

The LEIF framework also supports the integration of new LEIF Views, or user interface applications that can view data either generically or based on expected inputs from known LEIF Producers. A LEIF View uses the InfoBus to consume the display-filtered data from the display model. By default, LEIF Views are based on the Java JFC/Swing user interface toolkit and typically manage one or more JavaBean components to display data in various visualizations (geographic maps, data plots, time plots, tables, etc.). Data can be dynamically interrogated and displayed in a variety of views. LEIF Views can also be tailored to provide nonwindow services on the display-filtered data from the display model. For example, services could include data forwarding, which would forward the display model objects to external CORBA-based display applications that can also use the display information.

☆Bruce Eckel

☆Category: *Beginner's Book*

☆Winner: *Thinking in Java*

Bruce Eckel developed his book, *Thinking in Java*, during the creation and delivery of a number of incarnations of his public Java seminar. During this process he observed that the problems most people had involved language issues. His goal for the book was to teach the core language and to give people a solid foundation in what the lines of code mean before they jump into the use of Java libraries, such as for applets and windowing. He has found that once the fundamentals are understood, it's easy to acquire a new library. Without the fundamentals, there are constant roadblocks to developing that understanding. Consequently, it's not until Chapter 13 that applets and GUI programs are discussed. But by then, Eckel feels, people should understand the other issues and can focus on the libraries themselves. Readers will find that the programs are all designed to be compiled on the command line with the free JDK from Sun. And though they'll work with any conforming Java tool, the author has made an effort to avoid any vendor dependencies.

Eckel has created a system that automatically extracts code listings from the book for compilation, which reasonably ensures their accuracy. He put the book on his Web site (www.BruceEckel.com) and received a number of corrections from people, which he feels made a huge difference in its accuracy. This process also helps sell the associated CD-ROM, which contains all the slides and spoken lectures from his Java seminar.

☆EnterpriseSoft

☆Category: *Reporting Tool*

☆Winner: ERW Pro

EnterpriseSoft's Report Writer – Professional Edition (ERW Pro) for Java is a full-featured report writer. The core functionality of the software is to assist the user in creating a report definition file (called report templates), extracting data from a data source, and analyzing and outputing the data in a professional-quality report that's both informative and impressive.

ERW Pro is data-source–independent. It can extract data either from databases or from objects in an application. Most commercial databases such as Oracle, SQL Server, Sybase and MS Access are supported via JDBC or ODBC.

THE AWARDS

Application data is queried using an object-relational mapping architecture. Application data reporting is fast and efficient and can scale to tens of thousands of objects. It's ideal for distributed applications and is the preferred model of data access for enterprise-class applications.

ERW Pro provides multiple nested/sibling sections with dependent queries, nested groupings, a table of contents for viewing, charts/graphs, report executables and expression evaluations featuring user-defined function capabilities. The application data source uses an SQL-compliant querying functionality and can be extended to add one's own operators using user-defined query operators.

Reports in ERW Pro can be exported to HTML, PDF or

ASCII-CSV file formats. They can also be sent to the printer for WYSIWYG hard copy or simply viewed on-screen. Efficient report distribution is possible using EnterpriseSoft's proprietary DAT file format.

ERW Pro is easy to use. The GUI is simple and intuitive and works similarly to the GUI found in legacy Windows-based report writers. Queries are autogenerated based on the report template.

ERW Pro is designed with the Java developer in mind and is compact enough (about 500 KB; 350 KB for runtime) to be embedded into an HTML page or inside an application. Developers can use its API to insert this software into their applications. Licensing is developer-based with limited, royalty-free redistribution.

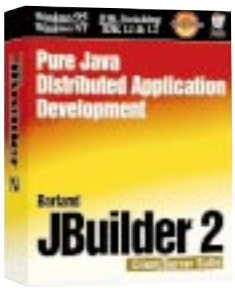
ERW is certified 100% Pure Java and can run on most platforms for which Java 1.1.5 or higher JVMs are available. Tested platforms include Solaris 2.5.1/2.6/2.7, Windows 95/98/NT 4.0 and Mac OS 8.0 or higher.

☆Inprise
☆Category: *Java Development Environment*
☆*Winner:* JBuilder 2

JBuilder 2 is a comprehensive visual development tool for creating pure-Java distributed enterprise applications. It includes a combination of features for creating platform-independent business and database applications, distributed applications and JavaBean components.

JBuilder 2 can increase productivity with a visual development environment that includes a component palette, fully integrated application browser, project manager, visual designers, Pure Java Two-Way Tools, code editor, HTML viewer, graphical debugger and a fast compiler. With the Pure Java Two-Way Tools, the code editor and visual designers are always synchronized, with no proprietary markers, macros or tags.

JBuilder supports the latest Java standards, including JDK 1.2, JDK1.1, JFC/Swing, Enterprise JavaBeans, JDBC, Servlets, Serialization, RMI, CORBA, Security, JNI and JARs. In addition, JDK Switching allows developers to compile against any JDK using JDK Switching.



Reusable and Enterprise JavaBeans can be created instantly with BeansExpress. JBuilder includes visual bean designers for Properties, BeanInfo and Events, making it easy – point and click – to create industry-standard JavaBeans. BeanInsight provides an analysis tool for diagnosing JavaBeans, including information on valid properties, property editors and Customizers. In addition, JBuilder has the largest library of JavaBeans, with 200+ components, including source code.

Corporate data can be managed with JBuilder's Pure Java DataExpress, which provides components for visual development of database applications using industry-standard JDBC connectivity. Master-detail relationships, Picklists, Lookups, MultiTable Joins and transaction processing can also be created easily.

The Inprise Deployment Server allows Java developers and information system managers to centrally deploy, manage and update their Java applications across corporate information networks. IS managers can reduce application deployment and maintenance costs while remote clients gain immediate, reliable access to the most current version of any application.

Scalable enterprise applications can be built with VisiBroker for Java. JBuilder 2 includes VisiBroker for Java 3.2, seamlessly integrated into the environment and project management systems so that complex CORBA applications can be delivered quickly. The Visigenic IDL compiler is invoked as part of the normal build process for a project, and automatically translates all IDL files in the project into the OMG-compliant Java binding.

☆InterNetivity
☆Category: *Reporting Tool*
☆Winner: dbProbe

For organizations that wish to deploy business intelligence capabilities to their users, InterNetivity dbProbe 4.0 delivers Web-based decision support that's easy to deploy and use. The product's interactive OLAP and standard reporting client allows Web-based users to access and analyze data from a variety of sources, including ODBC and Microsoft OLE DB for OLAP-compliant servers.

Administrators can deploy dbProbe in client-only or client/server mode. Deployment is straightforward in client-only mode; using a data source, administrators create a data cube and standard reports that are automatically published inside a single HTML page ready for deployment via the Web. There's no client software to install and no incremental effort required to scale up to thousands of users across an enterprise. Administrators can set up OLAP channels for distribution of updated cubes to their users. Larger data cubes remain on the server for remote access via the Web. For users it's completely transparent: they open a single HTML file from the Internet or intranet server that automatically downloads the dbProbe Java applet and then lets users navigate the data.

Written in 100% Pure Java, the dbProbe client lets users drill down, slice and dice, graph, create and share reports. Users can create new categories; hide, group and sort categories; and export data directly to their favorite spreadsheet or print it on any printer. The dbProbe client occupies less than 400 KB, supports mobile users and offers fast performance.

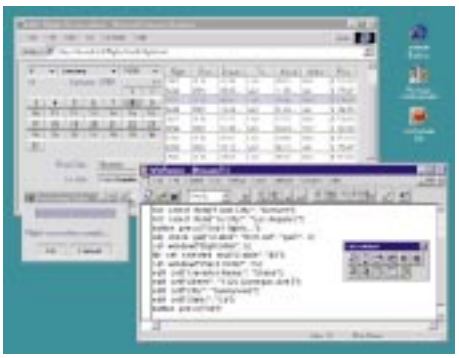
An economical, efficient way for organizations to distribute business intelligence tools to their Web-based users, its architecture makes dbProbe a good choice for software vendors looking to incorporate Web-OLAP capabilities into their tools.

☆Mercury Interactive
☆Category: *Testing Tool*
☆Winner: WinRunner

WinRunner is an enterprise-functional testing tool that verifies that Java applications are working as expected. By capturing and replaying user interactions automatically, WinRunner identifies defects and ensures that Java applications work flawlessly the first time and remain reliable.

The adoption of Java in mission-critical applications is causing a shift in the complexity of application architectures and in the client software distribution process. The need for testing in different environments occurs not only across different platforms but also between various browsers, virtual machines, windowing toolkits and releases of the JDK.

WinRunner simplifies test automation by approaching the task from a business-process perspective. While a user accesses the Java application, WinRunner automatically translates user actions into clear, readable test scripts that



can later be replayed to verify the functionality of the later builds of the application. It supports script enhancements as the application is developed or updated, executes scripts, reports results and enables script reusability throughout an application's life cycle.

WinRunner works with any Java-based applets or applications. With its sister product, XRunner, it offers full support for different hardware platforms (Solaris, HP/UX, Windows NT, 98 or 95), different browsers (Microsoft Internet Explorer, Netscape Navigator or Sun's Appletviewer) and different windowing toolkits (Oracle's EWT, Sun's AWT, Sun's JFC/Swing, Oracle's Developer/2000, Symantec's Visual Café and others). WinRunner test scripts can be leveraged across any combination of these environments, shared between Java, ERP application fronts and custom client/server clients, and even used for load testing. For example, users can develop a WinRunner test with Internet Explorer on Windows 95 and run it without any changes

with Netscape Navigator on Solaris.

Mercury Interactive's integrated testing solution for Java-based applications comprises the products TestDirector, WinRunner, XRunner and LoadRunner. These tools provide functional and load testing across the wide range of platforms, browsers and architectures found in Java implementations while leveraging test script usage between environments. WinRunner (for Windows-based applications) and XRunner (for Unix-based applications) automate functional and regression testing of Java clients. LoadRunner performs scalable load testing of Java-based systems while TestDirector organizes the entire testing process and manages high testing volume of Java-based applications.

☆ObjectSpace

☆Category: *Class Library*

☆Winner: JGL

ObjectSpace JGL – the Generic Collection Library for Java – is a Java adaptation of the ANSI/ISO standard template library that extends the JDK with a series of 11 advanced collections and more than 50 generic algorithms.

JGL is designed to complement, not replace, the basic features found in the JDK and is of particular use to enterprise Java developers. JGL includes full source code, hundreds of examples and a comprehensive online HTML tutorial and class reference. JGL enhances distributed collection support, allowing the remote construction, access and persistence of all JGL containers using ObjectSpace Voyager, the standards-neutral platform for object computing.

The JDK, which contains limited support for data collections and algorithms, was designed to provide a minimal subset of features used by the majority of Java developers. Although the JDK is sufficient for simple applet design and other such uses, JGL offers serious developers not only essential enterprise collections, but also the advanced data processing algorithms needed for use with those collections. These algorithms have been designed for use on JGL collections, Java-native arrays of primitives and objects, and all JDK collections. The generic JGL algorithms can be adapted using function objects and predicates to solve most collection processing problems.

Since June 1996, when JGL 1.0 was offered free to the Java community, this 100% Java, high-performance extension of the basic JDK features has been licensed by 10 major IDE vendors and downloaded by thousands of users.

☆Protoview Development

☆Category: *Best Bean*

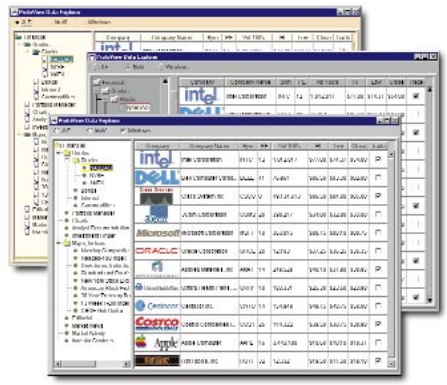
☆Winner: JFCDataExplorer

One of the first products to fully take advantage of Java Foundation Classes, the JFCDataExplorer is built on the familiar foundations of three JFC components – JTree, JSplitter and JTable – and Protoview has integrated them into a single component with added functionality and extended features.

Developers working with the JFCDataExplorer can take advantage of the synchronicity ProtoView has programmed between "panes" of the component. The left-hand pane contains a treeview that functions as a hierarchical structure for application data. Clicking on the nodes of the treeview allows the right-hand pane to populate with data from any source. While developers can default to the JTable data model, ProtoView has created a custom data model specifically for the JFCDataExplorer that incorporates a treeview into its logic and structure.

With each node click, the JFCDataExplorer retrieves data and populates the JTable with its corresponding column configurations. It also allows developers to add a column with images (the first column in the grid) at runtime. This column is bound to each row of data and moves when developers sort or move rows.

In addition to the standard JTable as the right-hand



pane, ProtoView has opened the JFCDataExplorer to accept any Java component or panel in that pane. Using the treeview to drill down on nodes, the JFCDataExplorer creates the perfect UI for organizing and streamlining application data. With limited screen real estate, developers can create data hierarchies with corresponding grids, charts, calendars and panels on a node-by-node basis. Like all ProtoView JFC products, the JFCDataExplorer takes advantage of the "pluggable look and feel architecture" of JFC, which allows it to easily switch from Motif, Java or Windows displays on the fly.

☆Rational Software

☆Category: *Modeling Tool*

☆Winner: Rational Rose

Rational Rose 98 provides Unified Modeling Language-based modeling for designing component-based applications. UML, pioneered by Rational and officially adopted as a standard by the Object Management Group, is the industry-standard language for specifying, visualizing, constructing and documenting the artifacts of software systems.

Rational Rose 98 features multilanguage capabilities and enterprise team development features, including integration with Rational's ClearCase software configuration management product. It provides a component-modeling approach to application development with support for COM, ActiveX and JavaBean components. The Enterprise Edition has multilanguage support that allows multiple languages to be mixed and matched within the same model.

Rose 98 supports C++, Java, Smalltalk and Ada, as well as 4GLs such as Visual Basic, PowerBuilder and Forte. For Java development, it supports the design, modeling and visualization of all Java constructs, including packages, classes, interfaces, imports, inheritance, fields, methods and modifiers. In addition, Rational Rose 98 can automatically generate Java source code and reverse-engineer Java source and byte codes. It also offers extensive support for object-relational databases such as Oracle8.

To enable development teams to share project information, Rational Rose 98 provides integration with ClearCase software and Microsoft's Visual SourceSafe version control software. Rose 98 models may also be published to and imported from the Microsoft Repository.

Rose 98 provides a framework library that contains templates with predefined components for modeling certain systems and includes frameworks for databases, the Internet and the Microsoft Transaction Server. Users can store their own models in the framework library and then make them available to other development team members.

Rational Rose 98 is available in three editions for Windows 95 and NT. The Enterprise edition provides multilanguage support for C++, Java, Visual Basic, Oracle8 and other languages. The Professional edition provides single language capability and is available for C++, Java and Visual Basic. The Modeler edition supports UML-compliant modeling. Rational Rose is also available for the following UNIX platforms: Sun Solaris, HP-UX, SGI IRIX, IBM AIX and Digital UNIX.

☆Riverton

☆Category: *Modeling Tool*

☆Winner: HOW 2.0 for Java

HOW is a component-based modeling tool and deployment framework designed for developers of distributed business systems. With the needs of business developers in mind, HOW extends the traditional idea of modeling to include development and deployment. HOW makes it straightforward for mainstream developers to build distributed and Internet systems in Java that comply with Sun Microsystems' Enterprise JavaBeans specification and allows these developers to leverage the growing number of EJB-based application servers.

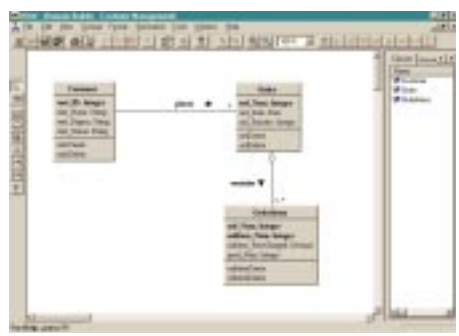
HOW gives Java developers a set of business analysis and component modeling tools that understand business systems:

- Requirements gathering is accomplished using HOW's Business Rule Builder.
- Developers define the business problem and its business process context in HOW's Use Case and Workflow Builders.
- They use HOW's UML-compliant Domain, Interaction and Activity Builders to design the system's business components, describe their interactions and graphically depict their behaviors, respectively.
- HOW extends UML to include the Task Builder, a graphical tool for architecting the system's visual components, and the Query Builder, another graphical tool for defining middle-tier business component data access.

From these models HOW users generate Java classes – including Enterprise JavaBeans – that form the basis for Java business applications. HOW generates:

- Design objects (class objects and domains) into Java classes, including Enterprise JavaBeans (session and entity beans)
- Queries into classes that implement JDBC-embedded SQL statements
- Class methods that allow convenient traversal of associations as collections of business objects

These HOW-generated classes can be loaded into a Java development environment and elaborated as required by the application. HOW is designed to integrate tightly with all of the major Java development environments, which ensures that the transition from HOW to IDE and back is seamless. HOW's ability to synchronize development and design environments assures round-trip engineering and



loss-free code generation.

Developers can also use HOW to preserve and reuse existing work. In addition, it can capture classes from data models or Java source files. Once in HOW, these classes can be modified or enhanced and then regenerated – even as Enterprise JavaBeans.

HOW is built on a multiuser object repository that promotes team development and creates a climate for component reuse. Developers share and reuse individual components. And HOW's integration with popular configuration management products means that versioning and archiving take place at the component and object level.

HOW-built Java components can execute on COM+ or EJB application servers, using either DCOM or CORBA as their distribution protocol.

☆Secant Technologies
☆Category: *Middleware*
☆Winner: Extreme POS

Secant Extreme Persistent Object Service for Java provides a Java-to-RDBMS and ORDBMS integration solution. Extreme POS is a powerful development and runtime environment that simplifies the assembly of the data-intensive business applications that integrate Java objects with enterprise-relational data. For three-tier development, Extreme POS provides an efficient in-memory caching and database connection-pooling architecture, making it a good choice for companies that are developing CORBA-based systems and want to use a scalable, persistent object service to support the data-intensive needs of hundreds of concurrent users. Extreme POS can also be used in a classic client/server configuration, allowing each client to have its own database connection.

Extreme POS provides a complete, ready-made infrastructure solution that supports both forward and reverse engineering. It will generate database schema and mapping code from an object model and will also generate an object model and mapping information from an existing database model.

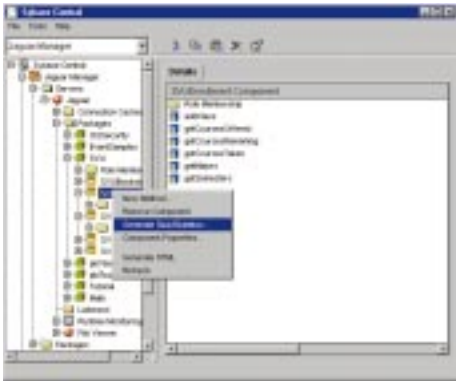
Extreme POS is unlike competing JDBC products that implement call-level interfaces. Such products force Java developers to work at the SQL level with rows and columns. As a result, developers must hand-code a specific Java-to-relational-mapping solution for each application. Because the SQL data mapping code is intermixed with the business

logic programming code, the resulting applications are difficult and costly to maintain. The JDBC approach is time-consuming, expensive and error-prone. Extreme POS eliminates the deficiencies imposed by other solution approaches and provides the following benefits:

- Productivity: Application development time is reduced to 20% - 40%. Developers write no database mapping code, as code is generated automatically from a standard object definition language or from a Rational Rose 98 UML description of the business object model (when used in conjunction with Secant's Rose Secant Extreme Link product). With 100% portable applications across all supported databases, no extra porting work is required.
- Performance: A ten- to a hundredfold improvement in application runtime performance is typical, as objects are read into memory only once, using an advanced two-level, in-memory caching architecture. Coupled with an enterprise scalable architecture, high-transaction volume applications are easily supported.
- Extreme POS's integration support is seamless and easy to use, and it makes a relational database look and perform like an object database. Developers perform all querying, navigating and updating in terms of the object model, not the database. No SQL is ever needed.

☆Sybase
☆Category: *Application Server*
☆Winnner: Enterprise Application Server

The Sybase Enterprise Application Server (EAServer) is a scalable deployment environment that supports simple, dynamic, data-driven Web sites as well as fully integrated, component-based information systems. EAServer includes a component transaction server (Jaguar CTS) and a dynamic page engine (PowerDynamo) to deliver rich, dynamic data publishing and OLTP for the Web. This provides customers with a complete platform and enables them to build the next generation of applications that call for stronger



relationships with back-office systems, such as customer self-service, dynamic information, trading networks and enterprise application integration.

EAServer provides both Web and nonWeb access to back-office systems, allowing customers to leverage their current IT investments by reusing existing skills and applications. It delivers a flexible foundation that can scale from simple dynamic Web sites to complex, data-intensive, transaction-aware applications in one integrated environment.

With this product, Sybase delivers an enterprise class platform for distributed and Web applications. EAServer provides high-performance capabilities, including page caching and scheduling, connection, and instance and session pooling. With EAServer, applications are secure because they have the support of industry-standard Secure Socket Layers, which secure access to operational systems.

EAServer offers a standards-based environment, giving customers and partners freedom of choice in building applications. It provides support for most major components, including Java and JavaBeans technology, CORBA, COM and C/C++. Upcoming releases will add native support for PowerBuilder components and Enterprise JavaBeans. EAServer also supports over 25 databases.

EAServer is an integral part of Sybase's development tools, offering the choice of PowerBuilder or Java component development. This provides a complete Web development environment with team and site management, HTML and script editing, component assembly, script debugging and automated deployment.

☆Tidestone Technologies
☆Category: *Grid Control*
☆Winner: Formula One for Java

Spreadsheets are one of the most commonly used technologies for collecting, computing and displaying data. The Internet is a vast and powerful resource for enabling communication between organizations, their employees and their customers. Formula One for Java 5.5.1 from Tidestone Technologies, Inc., enables users to merge these two technologies and make it easy for spreadsheet data and functionality to extend wherever it's needed.

Primarily, Formula One for Java is a 100% Pure Java spreadsheet application. However, Java developers can use it as a JavaBean while Webmasters can use it as an applet. Powerful as well as versatile, it can be used on a server or a client and in Web-based applications. It includes several robust, cross-platform features and, in addition to spreadsheet applications, developers and Webmasters can use it as a grid to present data in tabular format, to perform simple or complex calculations, to access data in a variety of databases, and to write and format spreadsheet data in a variety of ways.

As a JavaBean, Formula One for Java can be used in many of the Java development environments on the market today, such as Symantec's Visual Café, Sun's Java Workshop and Inprise's JBuilder. Formula One for Java includes a robust API of more than 400 properties and methods. It also connects to most databases through its JDBC support and is capable of writing spreadsheets to HTML. Its lightweight file size makes it possible to build server-based applications that are easy for users to download while supplying developers with the spreadsheet functionality they require.

Webmasters can deliver live spreadsheets in their Web pages by using Formula One for Java as an applet and as little as one line of HTML code. They can also load spreadsheets customized with their own formulas and formats. A good way to move Excel spreadsheets across the Web or a network, Formula One for Java's small download and fast calculations ease the implementation of live spreadsheets in Java-enabled browsers.

The product includes a Workbook Designer that can be used on any platform as a stand-alone, Excel-compatible spreadsheet application. Users can implement numerous data formatting options and utilize Excel-worksheet function syntax. Formula One for Java also delivers fast calculations, making it a good cross-platform solution for reading and editing Excel files.

☆Timecruiser Computing Corporation
☆Category: *Best Java Application*
☆Winner: Timecruiser

Timecruiser 1.5 is a Web-based collaborative time management and event publishing application. Using Timecruiser, enterprise or community users can manage and communicate their event information such as corporate events, departmental activities or project schedules via intranet, Web site or extranet. It enables individual users to maintain personal schedules in accordance with group and public events.

Installed entirely on a Web server, the Timecruiser application is launched easily from any browser. Time-

cruiser's architecture allows users to post, coordinate, promote, invite and RSVP group or individual events placed on calendars, enhancing the cohesiveness for groups. Data is instantly portable and users aren't tied to the corporate local area network. Administration of Timecruiser also resides on the Web and can be accessed within any Web browser. It provides LDAP connectivity for centralized corporate directory database or stores configuration information on a proprietary database.

By attaching Timecruiser's framework of calendar applets – Capplets – to an event, publishers may provide additional contents in multimedia fashion such as video, sound or animated graphics. Within a Web calendar environment, Capplets may be used to attach interactive forms for users to book event tickets or seats, to attach directions and/or agendas for meetings or to get a syllabus for a course. Timecruiser propels the Web from an online reference source to a working application environment. Its features include:

- Web-based cross platform scheduling and event posting
- Multiple secured access levels
- Event forwarding and in-box
- Easy event editing
- Joint calendar views
- Multiple time prospects
- Assistant
- Recurring events specification
- Ticker tape event highlighting
- RSVP
- E-mail integration
- Multimedia Capplets
- Event advertisement
- Group schedule coordinating
- Alternative HTML calendar view
- Event search
- Print
- Configurable GUI and functions
- Local alerts
- LDAP directory support
- File system or ODBC/JDBC databases
- Multiple server support
- API
- Online enrollment / ticketing for events

Timecruiser is good for community members who wish to publish events or share calendars securely, or who want to coordinate group schedules, meetings or resources such as people, facilities and equipment. Current Timecruiser customers are from various market sectors including education, corporation and the government.

☆Zero G Software

☆Category: *Installation Tool*

☆Winner: InstallAnywhere 2.5

InstallAnywhere supports virtually every Java-enabled platform, allowing developers to create a single universal installer that will run on any platform. It handles all the platform-specific details, and supports Windows 95, 98 and NT, Unix (including Solaris, Linux and others), Mac OS and OS/2. InstallAnywhere tailors itself to the user's system, such as setting custom icons on the Mac OS or making Win 32 registry entries.

InstallAnywhere has a Java-based installer engine that can handle installations with thousands of files. The SpeedFolders technology introduced in version 2.5 uses advanced compression techniques to create installers that are smaller and faster than other platform-specific installers.

InstallAnywhere's interface provides a six-step Project Wizard for quickly building installers. For more complex

installations, an advanced mode allows developers to install files to multiple destination locations or to set rules to install files only on specific platforms.

One functional feature, especially for end users, is the ability to create double-clickable application launchers. These LaunchAnywhere executables give Java-based applications true double-clickable, nativelike icons that can be placed on the user's desktop, in the Windows Start menu or in the user's home directory. LaunchAnywhere automatically locates the correct Java Virtual Machine, configures all runtime options, including classpath, and starts the Java-based application automatically without any setup by the end user.

If the client has no Java VM, InstallAnywhere can automatically install one as part of the installation, eliminating the need to download and install Java separately. The product is fully compatible with the latest Java VMs, including Sun's Java 2 and Apple's Mac OS Runtime for Java 2.1, in addition to supporting Java 1.1.

For basic installations or an enterprise Web-based deployment solution, InstallAnywhere is a useful tool for deploying Java software.

☆IBM

☆Category: *Java Development Environment*

☆Finalist: VisualAge for Java

VisualAge for Java is a Java application development environment for building Java applications, applets, servlets and JavaBean components. Version 1.0 offered developer productivity, ease of use and other features. Version 2.0, which is aimed at setting the standard for the development of enterprise Java solutions, allows for the quick extension of existing data, transactions and applications to e-business, enabling large teams of developers to manage their Java projects and build large-scale, high-performance server solutions.

Version 2.0 features a new high-performance compiler, connections to more enterprise systems, team programming support and exploitation of the latest in Java technology. The Entry Edition is free from IBM's Web site; the Professional Edition is the tool for power users and programmers becoming familiar with Java; the Enterprise Edition offers additional functionalilty to developers working in large teams, developing for multiple platforms or extending existing server data, transactions and applications to the Web.

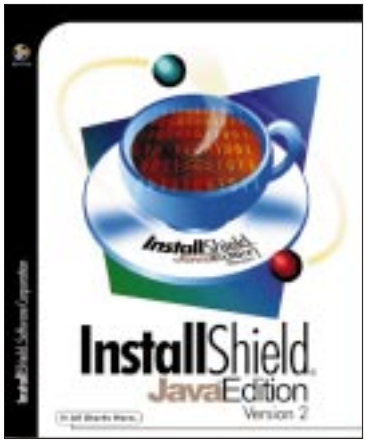
IBM's VisualAge Developer Domain offers developers a single access point for everything needed to build business-critical Java applications, including Java-related samples, education and support, as well as the ability to network with a community of Java professionals. Solution developers can also join IBM's Object Connection – Partners in Development program – to get advance copies of VisualAge for Java and to find assistance in marketing their products directly to VisualAge users.

InstallShield Software Corporation

Category: *Installation Tool*

Finalist: InstallShield Java Edition 2

InstallShield Java Edition 2, a 100% Pure Java product, provides Java developers with the ability to create setups that offer the InstallShield look and feel to end users regardless of target platform or operating system. InstallShield Java Edition 2 is easy to use. The development wizard walks developers through the installation process and uses the responses to its prompts to package the installation and application into a file that's readable by all operating systems supporting Java Virtual Machines version 1.1 or higher. Wizards guide setup creation, and InstallShield-Extensions, which customize installations, allow developers to plug in their own Java-based extensions. InstallShield Java Edition also supports silent mode installations and batch building via command-line access, and allows customers to run installations directly from a Web page. The InstallShield Installation applet enables users to specify the appropriate Java Virtual Machine for running the installation. If a Java VM isn't available, the applet can download one and install it for the user. The applet then downloads and installs the Java application via a browser link. Using InstallShield Java Edition 2 eliminates the need to create multiple installs for various platforms. The self-extracting Java class file created by InstallShield is universally understood by all platforms with specified Java VMs 1.1 or higher. This package contains a wizard that takes end users through a consistent installation process regardless of platform. Additional new features in InstallShield Java Edition 2 include:



- Self-extracting installations that embed installation and application in one file for easy launching
- Password protection for installations
- Improved compression and decompression
- The ability to create windows shortcuts anywhere on the target Desktop – Start menu, Programs menu, Startup folder, SendTo folder or directly on the desktop
- Launch scripts supply system properties from VMs as arguments
- On Windows systems, selecting Help automatically launches the Web browser
- GUI-based Extensions management

☆KL Group
☆Category: *Best Bean*
☆Finalist: JClass

Java Developers using the JClass Enterprise Suite of 100% Pure JavaBeans are creating professional GUI applications. Because all JClass components are database-aware, they bind automatically to a database, making RAD development easy. Developers working with the newly released Swing component kit and Java 2 (JDK 1.2) are now supported by the latest JClass releases.

JClass components support easy-to-use developer interfaces, offer a common API on all major JDK releases for easy migration and are supported in all popular IDEs. The JClass "family" consists of:

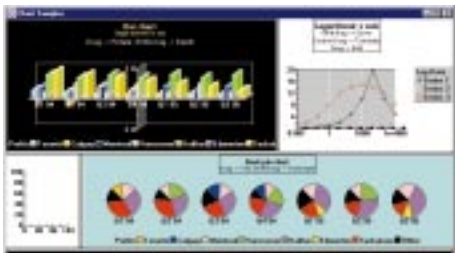
JClass SwingSuite: This set of extensions and enhancements for Swing in Java 2 (JDK 1.2) add the power of MDI, wizards and thread-safe, lightweight components.

JClass HiGrid: This RAD outline grid allows development of advanced multilevel data-bound database GUIs in minutes, in one component.

JClass DataSource: This robust, hierarchical multiple-platform data source provides the power of data binding to components and makes drill-down data access and updates trivial.

JClass LiveTable: This 100% Pure Java data-bound grid/table creates data-driven tables and forms with support for text, images and lightweight components in cells.

JClass Chart: This 100% Pure data-aware Java component enables quick and easy embedding of sophisticated graphs and charts into applications and applets.



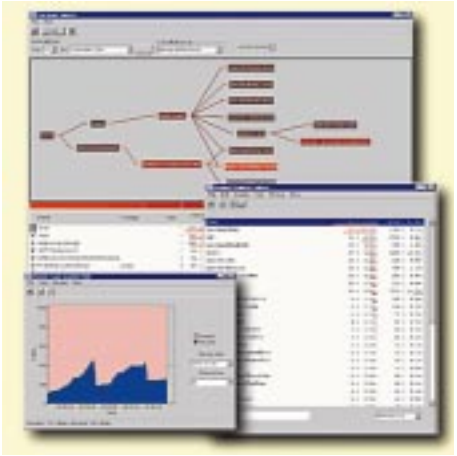
JClass Field: This 100% Pure data-aware JavaBean provides data input and validation for a range of popular data types with calendars, pick-lists, spin-boxes and combo-boxes.

JClass BWT: This collection of 100% Pure Java GUI components enhances and extends AWT to improve the look and quality of Java applications. The JClass Enterprise Suite of components comes with KL Group's Gold Support and Subscription, and includes the following features:

- Integrated, high-level 100% Pure Java GUI JavaBeans
- The ability to work within any JavaBean-compliant IDE
- The ability to connect to multiple levels of a master-detail relationship
- Robust database connectivity through JClass DataSource or through an IDE's data binding object
- JarHelper, a utility for creating a single JClass deployment archive that contains only the JClass products that are needed

☆KL Group
☆Category: *Profiling Tool*
☆Finalist: JProbe Profiler

Performance bottlenecks lurking in Java code can be searched for and destroyed quickly using the JProbe Profiler. The product provides accurate time and memory profiling with multiplatform support for Windows NT and Solaris in Java 2 (JDK 1.2), JDK 1.1 and JIT environments. Designed from the ground up for Java developers, JProbe Profiler's user interface and new heap reference tools find and eliminate memory leaks or simply explore how code runs. JProbe



Profiler comes with a Call Graph that enables developers to drill down to hotspots using nine performance metrics.

JProbe uses a standard Java VM licensed from JavaSoft and instrumented by KL Group. Thus JProbe can accurately measure virtually anything to do with Java program execution from within the VM itself. The instrumented VM combined with JProbe's graphical-based analyzer makes JProbe a powerful Java performance profiling and exploration tool.

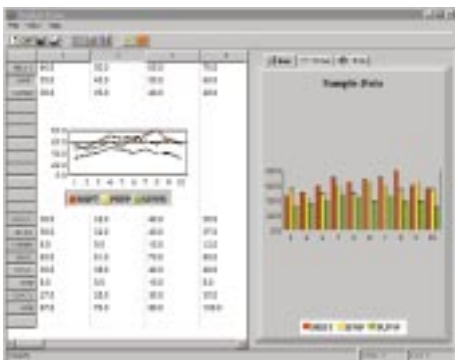
It also helps finish Java projects faster, and helps write code that's fast and memory-efficient. JProbe can eliminate inefficient algorithms, excessive object creation, I/O blockage, excessive thread creation, excessive method calling and inefficient memory usage. In addition, JProbe detects unwanted object references that are preventing objects from being garbage-collected.

Using JProbe's powerful GUI can teach developers things about Java code that are hard to learn any other way. Calling relationships and program structure – even for native methods and third-party code with no source code – can be probed. With JProbe, a program's memory usage can be watched while it runs, enabling developers to learn which operations are causing program size to grow or shrink.

☆Rogue Wave Software
☆*Category: Best Bean*
☆Finalist: StudioJ

StudioJ is a one-stop solution for GUI development, data analysis, charting and database access. The StudioJ suite integrates four libraries of pure Java components and classes from Rogue Wave Software and its Stingray divisions: Blend.J, Grid.J, Chart.J and DBTools.J.

- Blend.J integrates Rogue Wave's JWidgets and Stingray's Objective Blend to deliver more than 45 JavaBean components to Java developers. It provides both JFC and AWT support, with controls that offer the same interface and different implementations, rather than rewriting code. New controls include a splitter layout manager, regular expression filter and drop-down color chooser.
- With Grid.J, developers can turn a grid into a fully functional spreadsheet, with features such as automatic reference updates and circular dependency checks. Tree behavior can be embedded to give users an optional treelike view of a grid. Grid.J delivers both JFC and AWT support, with grids that offer the same interface and different implementations – again, without rewriting code. Developers can bind to any data source, including JDBC and ODBC. Support for DBTools.J is included when Grid.J is purchased as part of the StudioJ suite.



- Chart.J uses Model-View-Controller architecture based on the JavaBeans event model to provide for dynamic updates. (Charts automatically update as underlying data changes and the update interval or criteria are set.) With built-in callback mechanisms, users can drill down to display underlying data. Developers can bind to any data source, including JDBC and ODBC, and Chart.J includes support for JFC. Support for DBTools.J is included when Chart.J is purchased as part of the StudioJ suite.
- DBTools.J supports fine-grained error control and provides for flexible connection management. Just-in-time binding ensures streamlined performance since implementations are created only when it comes time to execute the object. DBTools.J has been optimized for applications using Oracle and Sybase, but also provides connectivity to any database that fully supports JDBC.

With integration support among the four libraries for seamless interoperation, StudioJ provides options such as tying a grid to a database and then feeding the data to a chart placed within the grid. Each library is also available separately, without the integration features.

☆Schlumberger
☆Category: *JavaCard*
☆Finalist: Cyberflex Open

The Schlumberger Cyberflex Open 16K smart card is built on the Java Card 2.0 API to provide easy application development, quick time-to-market and the most memory for Java-based multiple applications.

The Cyberflex Open 16K Development Kit features a PC/SC interface and fully integrates an application processor, a smart card simulator and a smart card manager.

Java-based smart cards allow applications from one or more providers to reside securely, side by side, on the same smart card. Multiple application cards can provide convenience for users and differentiated, market-specific products for issuers.

Smart cards, which incorporate a computer chip and memory instead of the traditional magnetic stripe, have been very successful in single roles – as bank cards, phone cards, electronic tickets and so on. But like early computer software, the programs they run are specific to a particular manufacturer's card and are for all intents and purposes "carved in stone."

Now issuers can put more than one application on a card, securely, and modify the applications after the cards have been issued, meaning that consumers will have cards that can perform a range of functions – such as debit, credit, e-purse, e-commerce and loyalty – whose applications can be changed and updated. This reprogrammable quality means that consumers will be able to individualize cards to reflect their own needs and priorities.

Schlumberger Electronic Transactions offers a flexible portfolio of smart-card–based solutions for businesses and communities of all kinds. The company provides cards, terminals, development tools and support in open configurations for operators, developers, integrators and distributors worldwide.

A Paradigm Shift in Distributed Computing

by **Bhaven Shah**

With the advent of Enterprise JavaBeans, Java server-side computing will be like never before

As Java takes a leap toward the next generation of enterprise computing, enterprises get ready to deploy large-scale business applications using Java. This article describes how the new Enterprise JavaBeans (EJB) technology from Sun Microsystems can be instrumental in building distributed enterprise applications. We'll first look at the application server implemented by Weblogic, which uses EJB technology to provide business solutions. Then I'll discuss how this powerful technology can be used in the context of a true distributed environment such as CORBA.

Background

Most of you are probably familiar with Enterprise JavaBeans by now. The EJB specification defines a server-side component model for development and deployment of portable, reusable, multitier and distributed Java applications. A detailed overview of EJB architecture and its components appeared in the September 1998 ***JDJ*** (Vol. 3, Issue 9), so I won't spend much time on that subject. But I will give you a brief overview of the EJB environment and its components before going into detail about the EJB-based application server and EJB's fit with CORBA.

Typical EJB Environment

As shown in Figure 1, a typical EJB environment can support a variety of client environments, including desktop clients connected through a LAN, Web clients, telephony clients, smart cards and other devices. Business logic is implemented as reusable application components deployed in the EJB application server. These components have complete access to all types of data and applications, including relational and nonrelational databases, flat files, live data-feeds and legacy applications.

Sun Microsystems' Java Platform for the Enterprise (JPE) provides a foundation for the development and deployment of Java-based enterprise-class application systems. EJB is an essential piece of the complete JPE environment that elevates Java to a serious application development language capable of supporting mission-critical distributed enterprise application systems. JPE API consists of nine Java APIs – EJB, Java Naming and Directory Services (JNDI), Remote Method Invocation (RMI), Java Interface Definition Language (Java IDL), Servlets and Java Server Pages (JSP), Java Messaging Service (JMS), Java Transaction API (JTA), Java Transaction Service (JTS) and Java Database Connectivity (JDBC) – that enable Java applications to access the core enterprise-class infrastructure services through a set of standard programming interfaces that include EJB components.

Some of the primary EJB components are the EJB server and container, enterprise beans, session objects, entity objects and EJBObjects. The function of an EJB server is to manage resources needed to support EJB components. An EJB container is where an EJB bean "lives." It provides a scalable, secure and transactional environment in which EJBs can operate. The container manages the life cycle of the enterprise bean. It also generates an EJB object that is an interface for the enterprise bean and represents a client view of the enterprise bean. All client requests directed at the EJB object (such as a request to insert transaction, state and security rules) are intercepted by the EJB container (see Figure 2).

The EJBObject acts as a proxy, passing method invocation requests to an instance of the enterprise bean installed and executing within the container of the EJB server. Clients use the JNDI and EJBHome interface of the enterprise bean to locate an enterprise bean.

Entity Beans vs Session Beans

Entity beans represent persistent data that's maintained in a database or any underlying data source. In contrast, session beans are created by a client and in most cases exist only for the duration of a single client/server session. An EJB container supplies a "factory" interface for creating new instances of the enterprise bean and a "finder" interface for locating existing instances of entity beans (see Figure 3).

Entity beans are transactional and can be recovered following a system crash. Entity objects are also implicitly persistent. They can either manage their own persistence or delegate it to their container. Entity beans usually represent a one-to-many relationship to client(s). In a typical enterprise some of the examples of entity beans would be employees, customers and orders.

On the other hand, session beans typical-

ly are neither transactional nor recoverable following a system crash. They can be stateless or they can maintain conversational state across methods and transactions. Session beans have to manage their own persistence. They usually represent a one-to-one relationship to client(s). Examples of session beans are electronic shopping carts and grocery lists.

What do enterprises need in order to develop and deploy an EJB-based application? First, they need to implement enterprise beans containing their business logic and assemble any third-party beans into their applications. Once they've done this, they can use any enterprise server implementation provided by vendors such as Weblogic, Persistence and EJBHome to execute and manage their EJBs. The server provider will take care of distributed transaction management, distributed EJBs and other low-level system services. Since Weblogic Inc. is an industry leader in the Enterprise JavaBeans server market and one of the early adopters of Enterprise JavaBeans, we'll discuss their application server in the following sections.

Tengah Application Server

Weblogic Inc., which announced its acquisition by BEA Systems as this article was being written, is the leading provider of Java application servers. Its first version of the Java application server – T3, short for three-tier – was shipped in the fall of 1996. T3 was also the first server in the market to support EJB specification. The name T3 then became Tengah. Tengah has Pure Java certification and currently conforms to the JDK 1.1. The Tengah application server logically integrates EJB, Web and transaction services in one server. With Tengah, users can easily and quickly build, as well as deploy, server-side business components (in the form of Enterprise JavaBeans). The following sections describe major components of the Tengah application server.

Tengah Enterprise JavaBeans

Tengah EJB claims to support all the features of the EJB 1.0 specification. It also supports entity beans with both bean- and container-managed persistence, as well as "automatic" persistence on files and relational databases via container-managed persistence. In addition, Tengah supports distributed transactions that permit multiple beans that are distributed across heterogeneous server machines to participate in a single transaction. Global naming with JNDI, RSI-based security and support for multiple invocation protocols such as CORBA IIOP, HTTP, HTTPS and TCP are also included in Tengah.

A unique feature of Tengah is the bean bar for Tengah beans. The bean bar is a collection of EJBs that provide access to Tengah services, including login, event subscribe, publish, Tengah server-side workspace access and more. Users can develop Tengah applications using drag-and-drop programming of Tengah beans in any JavaBeans-compliant IDE.

Tengah COM

Tengah also supports integration with the Microsoft COM. Tengah's COM compiler provides the means to build Java-to-COM bridges; Tengah RMI is then used to register these objects in JNDI and provide access to them from anywhere in the network. With Tengah COM on the client side, existing Java application components (like EJB objects) running anywhere in the network can be exported as COM business components (such as Visual Basic, Visual C++ and Active Server Pages).

Tengah Servlets

Tengah implements the session management API of the standard Java Enterprise Servlet model. Session management makes it easy to develop Web-based applications that need the server to remember the state of the client transaction across many browser-Web server (HTTPD) interactions.

Tengah Cluster

Tengah servers can be grouped into a cluster to enhance the scalability and fault tolerance of a Web application. Clustering can be used to spread requests from a large user community across multiple Tengah servers.

Tengah Zero Administration Client

Tengah's ZAC is designed to provide automatic distribution and management of application or system software throughout an enterprise network or the Internet. Once clients are registered with the ZAC, it automatically ensures that every client system is kept up to date without additional user intervention.

Tengah JDBC

Tengah provides a JDBC driver with connection pools for relational database access. A connection pool in JDBC allows multiple database connections at the same time. The connection pool can automatically shrink and expand depending on the load. This means that an expanded connection pool can gradually shrink to its initial size to release unused database connections.

EJB Development and Deployment Using Tengah

This section describes how to implement Enterprise JavaBeans as well as how to configure, deploy and manage these EJBs in the runtime environment of the Tengah EJB server.

Implementation

To implement an Enterprise JavaBean, it's important to get familiar with the EJB API provided by Sun Microsystems. (Please see the Resources section to download the EJB API.) Three main packages are shipped as part of the EJB:

- Package javax.ejb
- Package javax.ejb.deployment
- Package javax.jts

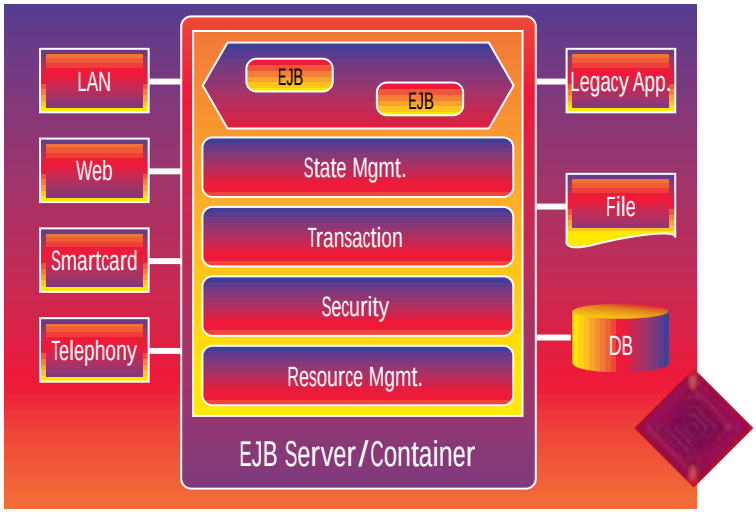


Figure 1: Typical EJB environment

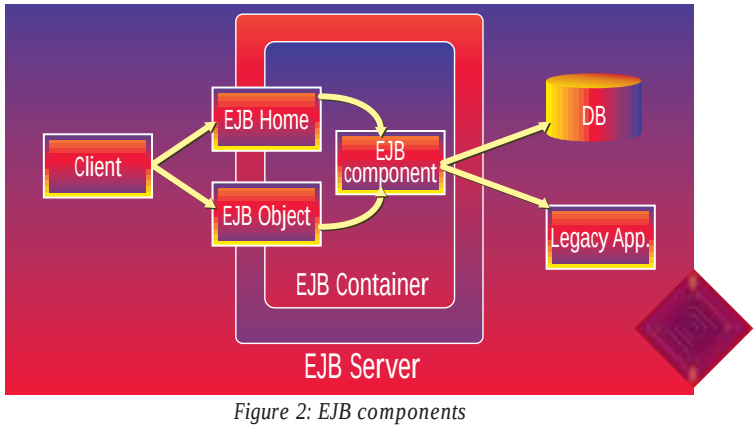


Figure 2: EJB components

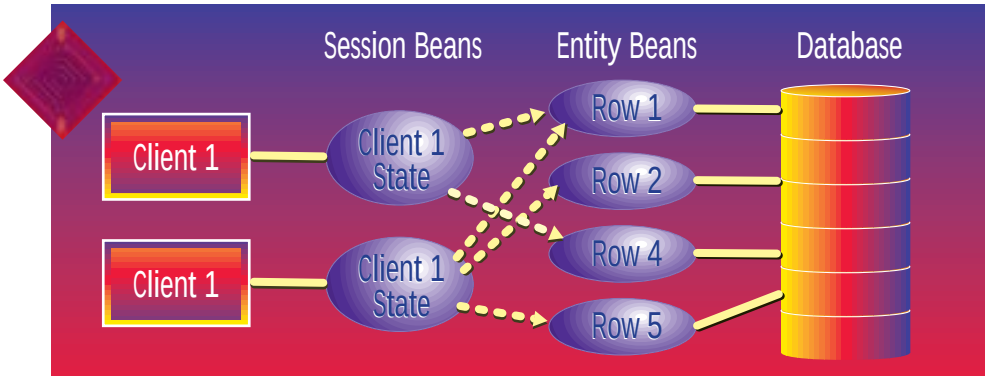


Figure 3: Entity and session beans

An Enterprise JavaBean is a class that a developer writes to provide application functionality. The developer can implement the bean as either a session bean or an entity bean by implementing the appropriate interface and declaring its type in the deployment descriptor used while deploying the bean. Typically, the developer will need to extend the javax.ejb.EnterpriseBean class from the javax.ejb package. Both the SessionBean and EntityBean interfaces extend from this class.

For instance, a session bean can be implemented as:

```
public class MyEJB implements javax.ejb.Ses-
sionBean….
```

And, an entity bean can be implemented as:

```
public class MyEJB implements
javax.ejb.EntityBean….
```

The methods in your Enterprise Bean will never be invoked directly from the client. The client uses the EJBObject, which acts as a proxy to call the bean's methods. The EJBObject is a "true" network object that gets accessed by clients over the network. You have to create a remote interface that will describe the business methods of the enterprise bean you'd like the client to be able to invoke. The method names and the signatures listed in this remote interface should exactly match those implemented in the enterprise bean. For example, you might provide a remote interface for the EJB as follows:

```
public interface Account extends
javax.ejb.EJBObject {
   public void deposit (double amount)
throws RemoteException;
   public void withdraw (double amount)
throws RemoteException;
   public double balance()
throws RemoteException;
}
```

It's important to note that the class javax.ejb.EJBObject will implement the java.rmi.Remote interface. The container vendor will generate the implementation code for the bean remote interface (Account interface in the above example) at the time the enterprise beans are installed.

For remote clients to access the enterprise bean, they first need to access the bean's "home" object which implements the home interface of the bean. The home interface lists one or more create() methods that can be used to create an instance of the enterprise bean. The server typically registers the home object (EJBHome) with the JNDI. The clients have to know the location of the namespace and the JNDI context factory class name for the home object. The home interface for an EJB bean might look like this:

```
public interface myHome extends EJBHome {
   public void create () throws  RemoteEx-
ception;
    public void create (String str) throws
RemoteException;
}
```

Again, note that the class javax.ejb.EJBHome implements the java.rmi.Remote interface. The container vendor is responsible for providing the home object that implements this home interface. The implementation code acts as a factory to create the enterprise beans.

Tengah EJB Deployment Wizard

Once you've implemented your enterprise beans, you can use tools like Tengah's deployment wizard to configure and deploy the beans for use with the EJB server within your enterprise. You can use this wizard to:

- Examine existing EJBeans and their deployment descriptor's configurable properties.
- Modify and persist EJBeans' properties.
- Generate EJBean interface classes for the EJBean execution environment.

You can invoke Tengah's deployment wizard from the command line with:

```
$ java weblogic.EJBDeployWizard options
```

In this instance, options can be used to get help or to set look and feel for the UI of the wizard.

Starting and Maintaining the Tengah Server

The Tengah server is a Java class file and can be started from the command line. Weblogic recommends that you start the server with –ms16m and –mx16m to ensure enough heap space for the server. This will ensure better performance in case of a more rigorous client connection load. The Tengah server also has its own garbage collection mechanism, purported to be more efficient than the one provided by Java. It might make sense in that case to turn off Java's asynchronous garbage collector. One possible command for starting the Tengah server could be:

```
$ java –noasyncgc –noclassgc –ms16m –mx16m
weblogic.Server
```

Weblogic also provides an administrative tool to start up, shut down and monitor the execution of the Tengah server.

Current State of Tengah

Weblogic is currently shipping version 3.1 of Tengah. It includes multithreaded implementation for RMI and network connection pooling. Other noteworthy features include clustering for fault tolerance using JTS and full EJB 1.0 support. It also claims to have COM component integration. In future releases Tengah plans to support integration with JMS and other messaging middleware such as Microsoft's MSMQ and IBM's MQSeries.

EJB Meets CORBA

To add to the story of paradigm shift in distributed enterprise computing, it would be worthwhile to mention that OMG is working hard to enhance CORBA to facilitate its adoption in the enterprise world. the CORBA 3.0 specification, already underway at OMG, will simplify the use of CORBA ORBs for the development of distributed object applications and include support for EJB specification. This will allow EJB components to be reusable as CORBA components. EJB architecture will be key to middle-tier solutions in CORBA-based applications. By treating EJB objects as CORBA objects, clients will be able to work with EJB components in different CORBA middle-tier environments without regard to the environment of the vendor hosting that component. A CORBA 3.0-compliant server, when encountering EJB components, will be able to load a

Java Virtual Machine at runtime and run the EJB objects. The first draft for the CORBA 3.0 specification was expected to be released by the end of 1998.

Products like Weblogic's Tengah are already providing the means to support CORBA objects within EJB containers. This is possible since CORBA objects can be wrapped as EJBeans within an application server environment like Tengah.

EJB and CORBA: Complementary, Not Competing, Technologies

CORBA provides a language-independent distributed environment, while EJB components enhance this environment with portable, reusable, multitier network-centric Java components. Thus these technologies should be viewed not as competing but as complementary technologies to provide a distributed platform for enterprise computing. The EJB model integrates well with CORBA's distributed object model.

One example of this is EJB's transaction model. EJB's transaction architecture closely models CORBA's Object Transaction Service (OTS) specification. The following section describes the EJB transaction architecture and its mapping to the OTS architecture.

EJB Transaction Architecture

An EJB runtime requires transaction propagation across multiple EJB servers on the network. This requires a distributed transaction service with two-phase commit to ensure recovery of transactions in case of a network failure. The EJB specification also requires the transactions to be able to span multiple EJB servers so that a bean in one server can act as a client to a bean in another server while preserving membership within the transaction. The transaction control for a bean is usually specified in the bean's Deployment Descriptor. The transaction could be bean managed or container managed. Various transaction controls can be specified for the enterprise bean: TX_REQUIRES, TX_NOT_SUPPORTED, TX-_BEAN_MANAGED, TX_REQUIRES_NEW and so on. Session beans can implement the SessionSynchronization interface to receive callbacks from the transaction service. The enterprise beans are transactional. They carry the transactional state in the form of EJBContext, which is provided to every bean when it's first created. The session and entity beans carry SessionContext and EntityContext, respectively, both of which are subclasses of EJBContext. In addition, an EJB vendor may choose to provide support for client-demarcated transactions. This would allow the client to begin a transaction prior to invoking a method on the bean.

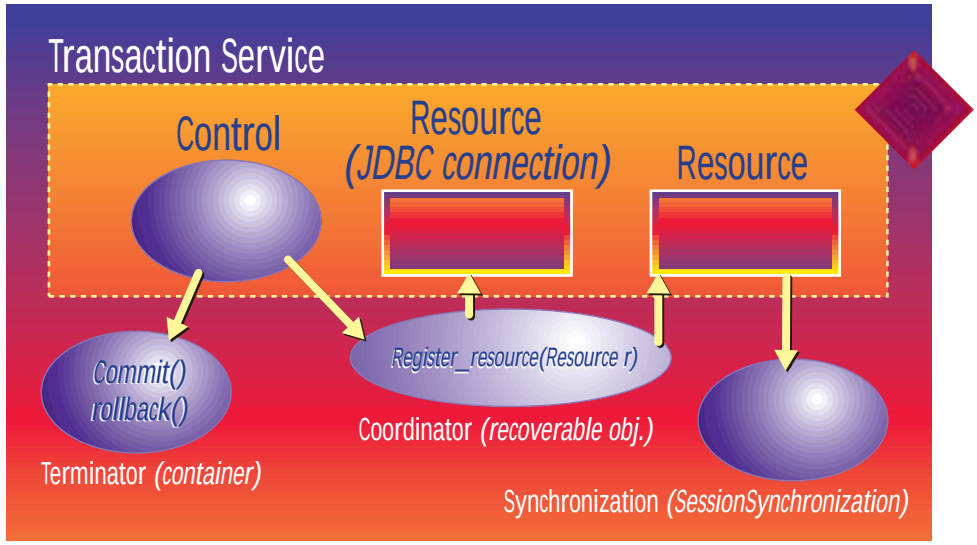


Figure 4: OTS architecture

Mapping OTS Components into EJB Components

The EJB transaction model is based on the OTS model specified by CORBA. In fact, the EJB specification requires its transaction service to be accessible using OTS 1.1 API. Hence, each of the OTS components can easily be mapped to an EJB object. For example, the Terminator object in OTS architecture could be viewed as an EJB container. In container-managed transactions the container can either commit or roll back a transaction, as appropriate. A transaction-aware JDBC driver in an EJB environment can act as a coordinator of OTS to register a connection with the transaction. Similarly, an EJB container can also act as a Coordinator to register a bean as a Synchronization object if the bean implements javax.ejb.SessionSynchronization interface (see Figure 4).

To keep the server requirements simpler, the current EJB specification doesn't require support for a nested transaction. Several other features of OTS aren't required by the current EJB specification, such as explicit transaction propagation. The JPE provides JTS, which is a Java mapping for the CORBA OTS. Enterprise JavaBeans support transactions using JTS. The EJB container and other transaction coordinator objects use the JTA to interface with the underlying OTS-compliant transaction service.

One other feature that I think will make CORBA a true counterpart of EJB is the addition of directory services to CORBA's naming service so that JNDI API can seamlessly access the CORBA service, or allow it to be integrated with any other namespace services.

What's Next?

The change in focus from client to server in the deployment of Java has created a demand for a component model that can target the server. Enterprise JavaBeans can satisfy this demand by providing a reusable server-side component model for the development of Java applications based on a multitier distributed architecture. The EJB component architecture represents a giant step forward in simplifying the development, deployment and management of enterprise applications. Enterprise JavaBeans will be a core piece of the JPE, which provides access to a core set of system services that's necessary for the development of enterprise-scale applications systems.

An adaptation of EJB by OMG would contribute significantly to its success. An interesting battle is emerging between Microsoft's middle-tier solution and EJB technology. Microsoft's distributed architecture, which consists of COM, Microsoft Transaction Service (MTS) and MSMQ is currently more stable and ahead in terms of implementation than the EJB architecture. The most recent acquisition of Weblogic Inc. by one of the veterans in distributed computing (BEA Systems) makes the battle even more interesting and indicates that the arrival of Enterprise JavaBeans may just be the beginning of a new era in enterprise computing.

Resources

OMG: www.omg.org
Weblogic Inc.: www.weblogic.com
Sun, Inc.: java.sun.com
Sun's Enterprise JavaBeans Page: http://java.sun.com/products/ejb
Sun's Java Naming and Directory Interface Page: http://java.sun.com/products/jndi

About the Author

Bhaven Shah, a member of the technical staff at i2 Technologies, Dallas, Texas, has BS and MS degrees in computer science. Bhaven has almost five years of programming experience with more than two years in Java. His current focus is client/server, distributed and GUI software development. Bhaven can be reached at bshah@i2.com.

bshah@i2.com

NetBeans Developer

by NetBeans

A Java-integrated development environment built completely within Java itself

by **Jim Milbery**

NetBeans
NetBeans, Inc.
Pod Hajkem 1
180 00 Prague 8
Czech Republic
Phone: + 420 2 8300 7322
Web: www.netbeans.com
Price: $149 / Developer

I recently had the opportunity to work with the NetBeans Developer IDE 2.0 for Java. Although the marketplace seems to be flooded with application development environments for Java programming, the team at NetBeans is offering a slightly different approach toward Java development. While the vast majority of Java-integrated development environments (IDE) are designed to run primarily under Windows-based operating systems, the NetBeans folks have built their development environment completely within Java itself. The result is a programming environment that runs on any platform that can support a JDK 1.1.5 environment.

NetBeans is based out of Prague in the Czech Republic and was founded in July of 1997. It is a privately held company with a number of high-profile investors, including Esther Dyson, CEO of EDventure Holdings. The NetBeans IDE has been designed around three basic concepts: functionality, platform independence and extensibility.

The NetBeans philosophy is that application developers need a powerful, robust IDE in order to successfully develop and deploy enterprise applications. Furthermore, the development environment should be able to support multiple platforms, even if the developer is primarily deploying to a single environment. Finally, the IDE itself should be extensible and support the ability to integrate new tools and features directly into itself as necessary.

Installation

I downloaded the NetBeans installation kit from their Web site, which is packaged as a single 7.7 MB Installshield program. NetBeans is offering version 2.0 of the NetBeans Developer, and you're free to download the software for trial purposes, non-commercial or educational use, free of charge. They're in the process of completing work on an Enterprise edition of NetBeans that will support some additional features, including the Cloudscape embeddable Java database due for release in early 1999. The installation went very smoothly and I was able to get the software installed and running in only 15 minutes. During installation the program searches your system for the location of your JDK 1.1 files. The system recommends version 1.1.7 of the Sun JDK, but I was able to select the 1.1.5 release that was already installed on my system without any problems. As part of the installation, the software configures the NetBeans development environment to use the JDK that you select during the question and answer phase. Although I installed the IDE under Windows NT, I looked through the newsgroups on the NetBeans site and on various other Java newsgroups for comments on installing the software on other platforms such as MacOS and Linux. In general, the product seemed to install reasonably well on all of the various platforms.

Using the IDE

NetBeans is a pleasure to look at, as you can see in Figure 1.

NetBeans is a Java application, which is one of the biggest differences between the IDE and other Windows-based Java devel-

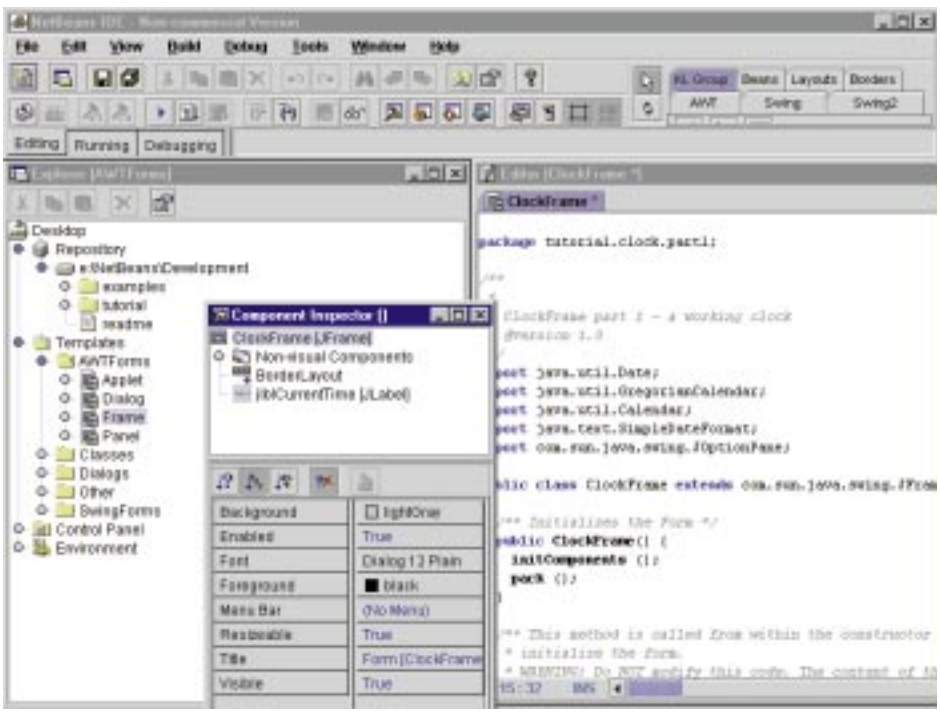


Figure 1: NetBeans' Metal style

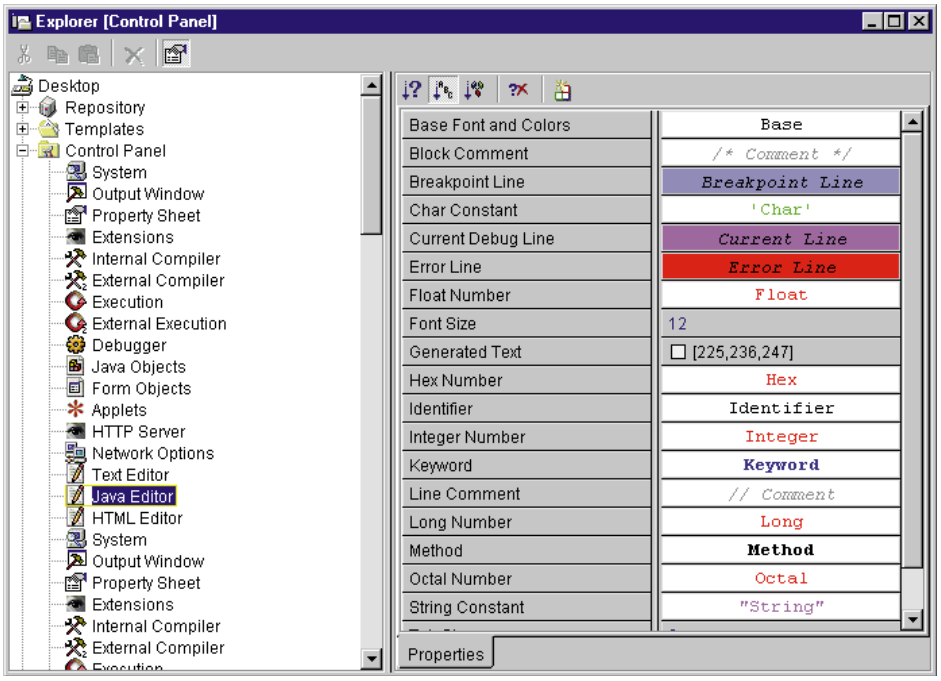


Figure 2: The control panel

opment environments. NetBeans is developed with the Swing JDK libraries and offers the crisp, distinctive look that you get with Swing. The IDE provides a menu that allows you to switch the look and the feel of the IDE to the various styles that are supported by Swing, including Metal, CDE/Motif and Windows; I chose to use the Metal style, as shown in Figure 1. One of the features of the IDE that NetBeans promotes is that it's highly extensible. Because the environment is written in Java, you can actually customize it to a larger degree than is possible with some of the traditional, Windows-based Java development tools. However, the IDE can be slow from a performance perspective. NetBeans recommends a P133 processor with 48 MB of RAM on Intel platforms for development. Although I was using a P200 with 64 MB of RAM and Windows NT4.0 SP4, the IDE routinely took over a minute to start up from the desktop. Also, navigating between menu choices and panels was sluggish, and I saw similar comments from developers in the newsgroups.

The desktop IDE is divided into a number of sections. The critical element is the Main Window, which is displayed as the top panel in Figure 1. The Main Window is well organized and I was able to get to most functions easily from the menus and icon buttons. At the bottom left of the Main Window panel is a set of three tabs that switch the focus of the IDE between the three various stages of development: edit, run and debug. If you're used to working with a multidocument interface style of development, you'll need to become familiar with the bevy of windows used by NetBeans. I wasn't able to find a way to mimic the MDI style of interface for the development environment. To make the process of managing so many windows easier, NetBeans offers the Workspace concept, which allows you to decide exactly which windows are open during edit, run and debug. I tend to want to see project information while I'm running and was able to use the Workspace technology to keep the Explorer window open during program execution. The Explorer window, shown on the left-hand side of Figure 1, organizes the various parts of the hierarchy, including the repository, templates, control panel and environment settings. The repository keeps track of all the file objects used by the IDE, and the templates provide ready access to blocks of functionality for your projects. NetBeans claims that their IDE is one of the most customizable interfaces on the market, and you can see some validity for this claim in the control panel, as shown in Figure 2. Between the control panel and the environment settings, I was able to tinker with almost every aspect of my development viewpoint.

NetBeans generates code for you and it also keeps track of code that will be regenerated so you never have to step over that code when you edit. You have complete control over how the source code will look in the edit window, including fonts and colors, as shown in Figure 2. In fact, NetBeans provides sets of preconfigured displays that I used to rapidly change the look and feel of the various editors. By virtue of these two panels you can customize all of the major components such as editors, debug windows and output windows to behave exactly as you wish, and this is part of what makes NetBeans interesting.

One of the nicer features of the product is the MultiWindow concept, in which all open editors appear as tabs from a single-edit window. I was able to use this feature to switch between editors, and you can undock any one of the editors as desired. There are numerous productivity enhancements that augment your development efforts, including the Connection Wizard, which can construct forms without requiring you to write Java code by hand. You can click from the source object to the target object and select parameters, methods and properties. NetBeans will create the Java code and display it for you in the MultiWindow editor. When you are ready to test out your code, NetBeans provides a lightweight, built-in HTTP server so you can test your applets with all of the constraints that'll apply when running from a remote Web server. Although the current version of the software doesn't support JDK 1.2, you can test JDK 1.2 applications by specifying a JDK 1.2 VM when you execute the application. The software ships with four tutorial applications that'll help you get acquainted with the NetBeans environment. I found them to be reasonable – but not exceptional – tutorials.

Summary

NetBeans is in the process of building an Enterprise version of the NetBeans Developer that will include support for features such as RMI, Enterprise JavaBeans and JDBC. This version will also include interfaces to version control software, which isn't provided with the current Developer Edition. I would recommend looking at NetBeans if you need to develop on multiple platforms or are looking for a less expensive Java IDE.

Test Environment

Client: Dell Pentium II 200 MHz, 64 MB RAM, 4 GB disk drive, Windows NT 4.0 (Service Pack 4), ViewSonic 15-inch SVGA monitor, 3COM Etherlink XL Adapter and 8X CD-ROM.

About the Author

Jim Milbery, an independent software consultant based in Easton, Pennsylvania, has over 15 years of experience in application development and relational databases. You can reach him at jmilbery@milbery.com or via his Web site at www.milbery.com.

www.milbery.com

Java Code Compilation

The "write once, compile anywhere" solution

by **Ajit Sagar**

In the November ***JDJ*** (Vol. 3, Issue 11) we peered into the Cosmic Cup to look at some of the Java Virtual Machines on the market. We also discussed how a VM enables Java to promote its "write once, run anywhere" (WORA) cause. To recapitulate, the Java programming environment 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-specific, executable instructions. A Java Virtual Machine's purpose is to load class files and execute the bytecodes they contain.

The speed of execution in the runtime environment is crucial for the success of the Java platform. After all, it targets several facets of the computing industry, but the crux of the platform is still the Java programming language itself. Several IT managers are putting off a wholehearted commitment to pure Java solutions because they still aren't convinced that it will meet the performance requirements for their applications. Happy programmers and cool languages don't put bread on the table.

This month we'll take a closer look at the two stages of compilation that lead to executable Java code. We'll also examine the available Java code compilation alternatives. Java is an interpreted language. An executed Java program typically consists of a virtual machine that interprets bytecodes. Interpreted languages can never be as fast as compiled languages because the process of interpretation consists of converting high-level programming instructions into machine instructions, one line at a time. On the other hand, natively compiled code is machine code – that is, the program is already in the form of machine instructions when it's ready to be executed. Obviously, the latter will execute faster.

WORA or WOCA?

Do we really need a "write once, run anywhere" solution? WORA essentially means that the deployed version of code is bytecode. The application is developed in the Java programming language and then compiled down to bytecode. This bytecode is shipped to the machine where the application actually needs to run. "Anywhere" requires that the same code should be capable of running on different hardware and operating systems.

What's the alternative to code that runs in a JVM? To achieve true native speeds for a particular platform, the application should execute native code (which runs as a process spawned by the operating system as opposed to code that's interpreted by a VM). This should be a no-brainer. Platform-specific optimizations can be performed most efficiently on native code. So how do we get native code from Java source code that runs on every platform? This is a bit of a dilemma.

The answer boils down to the responsibilities of the players in the application development process – the developer and the compiler vendor. If the developers end up using platform-specific compilers for individual platforms, they'll run into the very porting headache that Java has tried to eliminate. The responsibility of providing platform-independent executable code falls on the guys who write the compiler. If the same code could be compiled to different platform executables, we'd get the best of both worlds. This would be the "WOCA" – "write once, compile anywhere" – solution. The approaches taken by WORA and WOCA are illustrated in Figure 1.

Note: Neither WORA nor WOCA is feasible in a language like C or C++. The reason is that there are many platform-specific extensions of the programming language itself, which means that the source code written by developers for one platform will differ from the source code written for

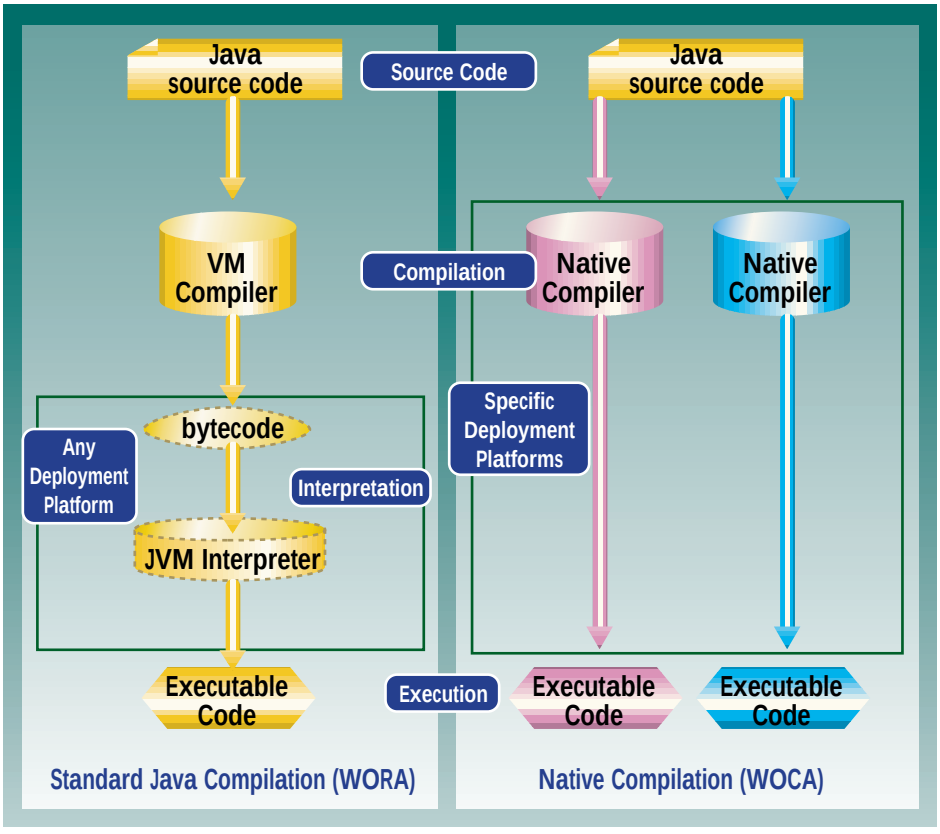


Figure 1: WORA vs WOCA

another platform. One of the greatest things Sun Microsystems has done for the computing world is to keep a tight rein on what goes into the Java programming language and to make sure that it has no platform-specific extensions. The result is that source code written in the Java programming language looks the same on any platform.

Compiled Just in Time!

Just-in-time (JIT) compilers translate Java bytecode into instructions that can be sent directly to the processor. A JIT compiler provides a second stage of compilation at the platform where code compiles the bytecode. Once recompiled by the JIT compiler, the code will usually run more quickly in the computer. Typically, JIT compilers come with the VM and their use is optional. The JVM passes on the .class file to the JIT compiler; the JIT compiler compiles them into native code for the machine that the application runs on. The JIT is an integral part of the Java Virtual Machine implementation. Since Java is a dynamic language, the bytecodes are "dynamically" compiled into machine code only when they're loaded.

JITs make the executable code faster only after it's called for the first time. In fact, the first time the class is used, it may actually be slower since it goes through an extra compilation step. Therefore, JITs are effective only when the code is used repeatedly. Going by the 80–20 rule (the program spends 80% of the time in 20% of the code), this happens most of the time and therefore JIT compilers speed up the execution considerably. In fact, when Java JIT compilers first came out, traditional Java interpreters were resulting in execution speeds that were 20 to 30 times slower than comparable programs written in C. JIT compilers brought the speeds of Java code from 40% to 60% of C/C++ execution speeds. Figure 2 illustrates the approach taken by JIT compilers.

JIT is currently the predominant technology for speeding up Java applets and applications. Almost all the compiler and IDE vendors – including Symantec's Visual Café, Borland's JBuilder, IBM's VisualAge, Asymetrix's Supercede and Microsoft's Visual J++ – provide Java compilers that have the JIT compilation option.

Find the "Hot Spots"!

While JIT compilers are a step up, they're still unable to reach the raw speeds of native code. Sophisticated optimization on JIT-compiled code is hard to achieve. Also, applying these optimizations slows the process of JIT compilation. For applications that require a larger performance gain, another viable approach is "adaptive optimization."

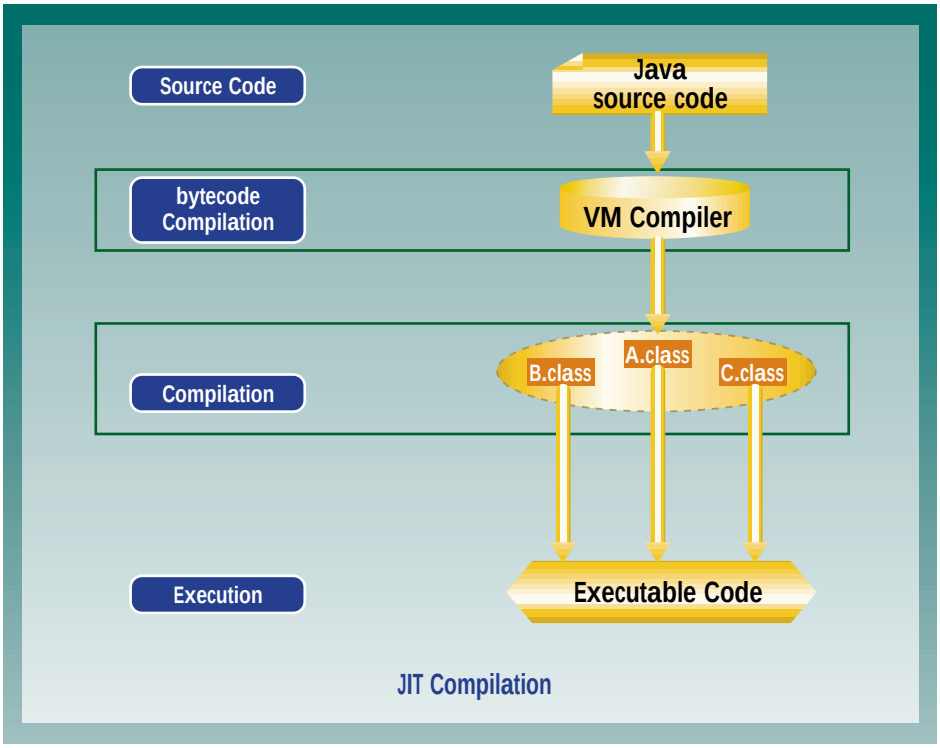


Figure 2: JIT compilation

Adaptive optimization further leverages the 80–20 rule. The logic that drives adaptive compilers is that since the majority of a program's executable time is spent in a small fraction of the code, it makes sense to concentrate on making that fraction as fast as possible. Adaptive compilation uses a more "dynamic" and "intelligent" form of compilation. The runtime system identifies the sections of code that are performance bottlenecks by continuously monitoring the executing code. It then applies sophisticated optimization techniques to speed up execution in these critical sections of code.

This technique is used by Sun's much-awaited HotSpot Virtual Machine. Fundamentally, HotSpot is based on compiler technology that is an extension of the JIT compiler technology. The HotSpot VM constantly monitors the performance of the executing bytecode. It "inlines" (remember the C++ "inline" keyword?) methods in the critical region for maximum performance. Inlined methods are like static methods in the class and are replaced by actual machine instructions (instead of method calls) in compiled code. Optimization in the HotSpot compiler is achieved with the help of a very advanced garbage collector and a new thread synchronization mechanism. Sun claims that HotSpot will make Java execution speeds comparable to C/C++ speeds. The compiler is expected to be released early this year and is more than a year late.

Native Compilers

JIT compilers and adaptive compilers are very important for improving performance on the client side, where the program may be deployed on various platforms. Applets execute by downloading bytecodes (.class files) from specific URLs. Thus the Java source code is compiled into bytecodes at the machine corresponding to the URL. These bytecodes may be downloaded to a multitude of clients – that is, to several platforms. Hence, native optimizations can't be performed before the code is deployed. JIT compilers apply optimizations that aren't really platform specific (as they target several platforms). They're also limited by the amount of time sacrificed in performing the compilations on the fly. An additional disadvantage of compiling bytecodes at the deployment machine is that decompiling them isn't easy. Therefore, companies can't prevent software piracy of their code.

The same restriction doesn't apply to code deployed at a single machine. In a client/server architecture this would be the server. Hence, code deployed at the server can be compiled down to native code, and global native optimizations can be applied to it. The trade-off is that this code won't run on other platforms. The reality is that it won't need to.

Native compilers compile Java source code down to platform-specific executables. In that sense they're no different from the traditional C/C++ compilers. As mentioned earlier, the unique feature that Java brings to the table is that the source code is always the same. Most IDEs provide the

option of compiling Java source to native code. However, most IDEs support only one platform. For example, JBuilder, Supercede, Visual Café, Visual J++ and so on run only on NT. As a result, although you develop platform-neutral source code in Java, you end up deploying on a single platform when you use a traditional native compiler.

This points toward the need for a cross-platform Java compiler, as illustrated earlier in Figure 1. In other words, we need a compiler from a single vendor that can compile Java source code down to native code, and that can take advantage of the specific platform it runs on. Such a compiler should also support compilation of bytecodes to support Java's dynamic nature. Currently, TowerJ from Tower Technologies provides these features. It provides support for several platforms, including NT and Solaris. Another cross-platform native compiler under development is JOVE from Instantiations.

What About Legacy Code?

Contrary to popular belief, the programming world will never be all Java. C/C++, Pascal, BASIC, Fortran, COBOL, Smalltalk and others will continue doing what they do best. Sun Microsystems recognizes this and that's why the Java Native Interface (JNI) is a part of the Java VM specification. Any vendor that provides a Java VM needs to support JNI (Microsoft thought they were the exception, but were recently proved wrong). JNI allows Java applications to interact with legacy applications written in other languages. Using JNI, the application or module is linked with Java code as a kind of shared library. If your Java application needs to interact with legacy code, you need to make sure that the compiler you choose supports JNI.

> ***"The speed of execution in the runtime environment is crucial for the success of the Java platform."***

Cosmic Reflections

Application development in Java is eventually going to require a mix of the various compiler technologies available. If speed of execution isn't a major concern, a traditional Java compiler may be used. If the performance boost provided by JIT compilers suffices, that option can always be turned on. If HotSpot lives up to its expectations, developers and managers for realtime applications can breathe easy. And if it doesn't, native compilers will grab a larger market share. On the server side, native compilers will always find a home. Whatever compiler technology is chosen for development, one significant service that Java has performed for the developers is that it has shifted a large part of the burden of supporting cross-platform applications to the compiler vendors.

About the Author

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

Ajit_Sagar@i2.com

Programming with I/O Streams: *Part 2*

Learning a few practical uses of Java's I/O streams is well worth your time

by **Anil Hemrajani**

*Last month's issue (**JDJ**, Vol. 3, Issue 12) covered the basic concepts of programming with Java's I/O streams, such as the difference between byte and character streams, the various stream classes, the concept of stream chaining and more. We'll conclude the subject this month by looking at some practical uses of these streams.*

There are so many uses of streams in Java apps that it's almost impossible to imagine all the ways they can be applied. Nevertheless, I'm going to show you several miniature programs that demonstrate some of the ways that Java API streams can be used. We'll also look at a sample (yet robust) application that uses a mixed bag of these classes.

File Input/Output

File programming in Java is easy -- there's a class to open a file for reading and another for writing to an output file. Listing 1 demonstrates a simple copy utility (similar to the MS-DOS "copy" and Unix "cp" commands). The code in Listing 2 initially opens the input and output files, then uses the following code to copy the input file to the output file and keeps a counter for the number of bytes copied:

```
    while ((read = fr.read(c)) != -1)
{
    fw.write(c, 0, read);
    total += read;
}
```

Properties

Java provides the java.util.Properties class as a facility for working with configuration files similar to the Windows INI file format that contains unique key=value pairs per line as shown in this example:

```
    user=anil
email=anil@divya.com
webPage=http://divya.com/people/anil/
```

Not surprisingly, the Properties class provides methods for reading from and writing to configuration files. Listing 3 demonstrates how to load a file using the Properties.load() method and how to extract parameters from the configuration file using the getProperty() method. To save to a configuration file, simply use the Properties.save() method.

Executing Programs

There may be times when you need to invoke command line programs (e.g., Unix sed or ls), pass data to them and read their output. To do this, you have to use the java.lang.Runtime class to start a program and communicate with it via its input, output and/or error streams. The following lines from Listing 4 run the Unix "ls -l" command and obtain the command's input stream in order to read the program's output:

```
    Process p      =
Runtime.getRuntime().exec( "ls -l");
InputStream is = p.getInputStream();
```

Servlet Programming

If you haven't worked with servlets and are still programming using CGI scripts, you're missing out on some cool stuff. Servlets are the server-side equivalent of applets (applets without the GUI). They also make extensive use of input and output streams (or *readers* and *writers*) as demonstrated in Listing 5. The following code fragment from Listing 5 reads all the input sent by the client (most likely an HTML form in a Web browser) using the POST method and echoes it back to the client using the output stream:

```
    InputStream  is = req.getInputStream();
OutputStream os = res.getOutputStream();
...
while ((c = is.read(b)) != -1)
   os.write(b, 0, c);
```

Object Serialization

Object serialization allows a developer to save an object and all its nontransient data to an output stream, and then restore it by reading it back in from an input stream. For example, if we have an "Employee&" object that contains three data members – an employee ID, name and salary – and we wanted to save objects for each employee (e.g., John Smith) to a file, we would use code similar to this:

```
    Employee e = new Employee(1, "John
Smith", 55000);
FileOutputStream f = new
FileOutputStream("JohnSmith.dat");
ObjectOutput s = new ObjectOutputStream(f);
s.writeObject(e);
```

To restore the object saved above, we would use code similar to this:

```
    FileInputStream in = new FileInput-
Stream("JohnSmith.dat");
ObjectInputStream s = new
ObjectInputStream(in);
Employee e = (Employee)s.readObject();
```

Note that the objects that need to be serialized must implement the Serializable interface by using the "implements java.io.Serializable" statement. However, this interface doesn't require any methods to be defined in the implementation class.

Incidentally, RMI (Remote Method Invocation) makes extensive use of object serialization to exchange objects between the RMI client and server.

Working with Streams in Memory

Working with streams in memory is just as easy as working with any other kind of streams. For example, if we modified the program in Listing 1 (Type.java) to work with a java.io.StringWriter instead of a PrintWriter, we could rewrite that code as follows:

```
    StringWriter sw = new StringWriter();
...
while ((read = fr.read(c)) != -1)
    sw.write(c, 0, read);
...
System.out.println(sw.toString());
```

Figure 1: Sample app using VariousStreams

Standard Input, Output and Error Programming

Standard device programming can be accomplished via three static data members of the java.lang.System class: System.in, System.out and System.err. System.out and System.err are of type java.io.PrintStream while System.in is simply a java.io.InputStream. We've already seen a couple of examples of System.out in our listings. Note that, like other classes, these too can be chained to achieve the required goal.

Sample App Using Socket, GZIP, File and JDBC Streams

Now that we've looked at several miniature programs, it's time to examine a more robust application that uses a mixed bag of the stream/writer classes.

Figure 1 provides a global view of the hypothetical application and is intended to demonstrate how a few lines of Java code can be empowered if the proper TCP/IP network is in place.

A hospital in California and another in Florida connect to their processing center (via sockets) in Virginia so they can relay information about their patients on a daily basis. The protocol used to send the data is simple: the first line contains the hospital name, the second line contains a patient name, and the the remaining lines contain the patient's data (e.g., name, address, medical history).

The complete code for the client-side processing can be seen in Listing 6.

```
<p align="center"><img
src="../../../~BUSIN~1/~PUBLI~1/IOSTRE~1/figure2.gif" border="1"
width="500" height="380">
<p align="center"><strong>
```

The server in the Virginia data center receives the client (California, Florida) data, stores a local copy in a file using GZIP compression, then forwards the same (compressed) data to a medical reporting agency in Europe by using JDBC to store the data in their relational database. Listing 7 shows the complete source code for the server-side processing.

The reporting agency then runs a program (see Listing 8) to view the patient data stored in their relational database.

Note that I haven't explained the programs in Listings 6 through 8, hoping that by now you've learned enough about Java streams to follow the code with the help of my comments in the source code.

Writing Custom Stream Classes

Writing your own stream classes is easy – you simply extend the top-level classes (java.io.Reader/Writer or java.io.Filter*) and implement the required methods. To get an idea of how to write your own classes, take a look at the source code for the various descendant classes in the JDK software. If you're using an IDE instead of Sun's JDK to develop Java programs, look around in the IDE's software directories for the JDK source code (e.g., \VisualCafeDbDe\Java\src).

Why would you write a custom class? Most likely because you want to process the data in a certain way when reading or writing it. Take our backup software, BackOnline, for example, which uses 56-bit DES encryption streams when backing up or restoring data. We had to write it because we wanted to use stream chaining and were missing the encryption stream classes since they didn't exist in JDK 1.0.

Summary

By now you should have a good handle on how I/O streams work. Keep in mind that many new APIs (e.g., Media, 2D/3D, Mail) not covered in this article make use of streams. So take a few minutes to explore the various classes in the java.io package and get a thorough understanding of them – it will be well worth your time if you plan to program in Java.

About the Author

Anil Hemrajani is a senior consultant at Divya Incorporated, a firm specializing in Java/Internet solutions. He provides consulting services to Fortune 500 companies and is a frequent writer and speaker. He can be reached at anil@divya.com.

anil@divya.com

SYS-CON RADIO INTERVIEW

Broadcast live at December's Java Business Expo in the Jacob Javits Center in New York City, SYS-CON Radio's Chad Sitler spoke with Ethan Henry of KL Group and Gregory Prokter of Slangsoft about their new products.

ETHAN HENRY of KL Group

JDJ: *Welcome to SYS-CON Radio's live broadcast from the Java Business Expo. Joining us is Ethan Henry, Java evangelist for KL Group. Thanks for coming today.*
Henry: Thanks, Chad.

JDJ: *There are a lot of companies here at the Expo making announcements, debuting new products and such, and I'm informed that for KL Group it's the same thing.*
Henry: Yes, we have a couple of announcements. We have a new release of our JClass line of JavaBeans, and we have a new release coming up very soon of JPro Profiler. Why don't I tell you about them?

JDJ: *Great, go ahead.*
Henry: So the new JClass 3.6 release – these were the first JavaBeans we released and were the first set of JavaBeans that are JDK 1.2- or Java 2-compatible. We were on the Net with an electronic download of our Beans within 20 minutes of the JDK 1.2 release – we had them all wrapped and ready to go. And as soon as Sun put JDK 1.2 up for download, we had our stuff up as well. We're pretty proud of that, and it was a real achievement. It was really great. The whole JClass product line is JClass Chart, JClass Live Table, JClass Field and JClass High Grid, and now they're all in versions that are Java 2 Swing-compatible. And we still have versions for the previous versions of Java for JDK 1.1 and even JDK 1.2.

JDJ: *Can you tell us a little more about the involvement with Java 2?*
Henry: Well, we actually have a whole new next generation of JClass JavaBeans coming out specifically designed for the Java 2 platform. The first one we've announced is Swing Suite. Swing Suite is specifically aimed at the Java 2 platform, and it aims to extend, enhance and provide additional components for Swing developers giving them more capabilities to build on top of what they already have inside the Swing component library that's part of Java 2.

JDJ: *Now if I'm a Java developer, and I'm going out there looking for a product like yours, why am I going to choose yours over someone else's?*
Henry: Well we're working closely with a lot of the developers at Sun inside JavaSoft who are doing things like the Swing development, and we really think we've got THE most complete solution of JavaBeans that you can get in one packaged family on the market. There are other companies that offer individual Beans like charts or tables or things like that, but we're the only supplier who gives everything in one integrated package with common APIs.

Additionally, we're working very closely with a lot of IDE vendors. So, if you're using a development environment like JBuilder, Visual Café or Visual H for Java – the pure Java development environment – we're working very closely with them and we have a great integration. JClass JavaBeans are a great accessory for any IDE-based developer.

JDJ: *Okay, so how about the products you debuted? I wouldn't mind talking about JPro.*
Henry: JPro Profiler 2.0 is another product we just announced, and again, it's offering support for the new Java 2 platform which is really great. Not only are we offering that support, but we've also added a whole slew of new features. We're not just looking at providing incremental upgrades every time the Java platform changes, we're looking at providing great new functionality for developers in all our products. So it adds support not only for time profiling but also for memory profiling. Developers who are having problems with memory usage can now examine objects, see what kinds of objects they have and see how much space they're taking up. This is really great functionality and you're getting information that you can't find any other way without using a tool like JPro Profiler.

GREGORY PROKTER of Slangsoft

JDJ: *Welcome back to the Java Business Expo on SYS-CON Radio. I'm joined by Gregory Prokter. He's the COO of Slangsoft. Welcome to the show.*
Prokter: Thank you very much.

JDJ: *Now I know you just announced the release of your two new products, Emule SDK 1.3 and Slang Suite 2.0. Can you get into those a little bit and tell us about them?*
Prokter: Yes, absolutely. The Emule SDK basically is an easy-to-use localization capability that delivers a flexible, straightforward and inexpensive Java localization solution and that can truly be defined as write once, support all languages and run anywhere. We've just announced the release of 1.3, which, on top of all the features that we had in 1.1 and 1.2, supports JavaBeans and Swing, and provides some additional capabilities and features for our customers.

JDJ: *So why would someone choose your product over another like it? What's the difference?*
Prokter: Well, the Emule SDK, which is our core technology, can localize your Java applications within minutes and it can work with either a JDK 1.0, 1.1 or 1.2 or any other JDK from vendors like Microsoft, Borland or Symantec. It provides support for 41 national languages. And the key concept here is that this SDK doesn't need any special fonts or native operating system language support, and no keyboard drivers need to be installed. So since the product is basically calculated for Java developers, all they need is to install the kit. Then they can begin programming in Java using the GUI provided by the SDK to enable support for 41 languages in the Java program.

JDJ: *Earlier you mentioned the Slang Suite. I know you're very proud of that product, so can you tell me about SlangMail and other components of the suite? Could you tell us what makes you so proud of it?*
Prokter: We got so excited and proud of our core technology, Emule SDK, that we decided to build something equally nice on top of it. Slang Suite included four ready-to-use shrinkwrap business programs that do daily tasks like browsing, chatting, e-mailing and word processing, and does it for 41 national languages. In other words, running under Windows 95, an end user could send an e-mail in Japanese or Italian while receiving an e-mail in Russian or Hebrew. So, this e-mail we call SlangMail is really unique because it can handle all the languages, is written 100% in our Java and doesn't require a recipient to have that SlangMail package in order to view the foreign text. It uses a very special technology. It's stand-alone technology that enables the recipient to actually view messages in a foreign language. That viewer is connected to our server and from that server the necessary fonts and input methods are downloaded. That concept enables the recipient to see the message in its original language without actually using any localized operating system support, without having installed any fonts or input methods – basically without any worries or even having to think about it. That's what makes this package so unique, and I believe it makes it must-have software for any international user or user who speaks more than one language.

JDJ: *You've mentioned some of your products and how innovative they are, but unfortunately technology changes; therefore, what's in the future for Slangsoft?*
Prokter: Well, we're looking to expand on the success of Slang Suite and to continue to add more languages to our end user applications. I know we're looking into adding Devanagari for our Indian-speaking users, and we're looking to get more Asian languages into our core technology and end applications as well.

We're looking to add more capabilities to our word processor and to do a complete integration of all the tools, thereby creating a complete communication center for the end user. That center will provide end users with the capability to do e-mailing, chatting, browsing and word processing in any national language and it's all going to be integrated.

About the Author

Chad Sitler, host of SYS-CON Radio, is SYS-CON Interactive's senior Web designer. He also runs a Web design company, Logos Web Design. Chad can be reached at chad@sys-con.com.

Java Developers Journal

SYS-CON Interactive

Java Developers Journal

SYS-CON Interactive

Java–The Software Design

Anybody can build an application, but can anybody design one?

by **E Ming Tan**

It's true that you don't need a computer degree to know how to program. However, to do it with the kind of quality that allows for easy maintenance and change is another matter. As we all know, based on Software Engineering (SE) principles, a software product's life cycle consists of analysis, design, implementation and testing. Most people seem to spend the majority of their time on the latter two phases while forgetting about or putting less emphasis on the former two. This is wrong, especially in the implementation phase. Implementation should constitute the smallest part of the whole development time. Part of the reason why programmers have to spend so much time and effort on the implementation – and then later on the testing – is from not having a good design in the first place.

In this article I'm assuming that the requirement analysis phase of software development is fully understood so I can focus this article on the design of software products.

According to object-oriented analysis and design (OOAD) in SE principles, objects in a system should exhibit the two most important properties: being highly cohesive and lowly coupled. There are various ways to design a highly cohesive and lowly coupled object in Java. The way that I'm sharing with you in this article is based on the Java 1.1 Delegation Event Model.

I'm naming the objects *radio broadcaster* and *radio listener*, based on their roles in the design. The reason I'm calling them radio broadcaster and radio listener is because of the similiarity between how these objects work and how radio broadcasting stations play a role in our daily lives.

For example, in this country (or, *within a software product*) anybody (or, *an object*) can set up a radio station and become a radio broadcaster. In this way, the radio broadcaster (or, *object*) becomes a source of information (or, *becomes the provider of new service for the system*) for the listeners (or, *objects*) in need of that information. Since in real life radio programs change every day, even though the interests of radio listeners tend to be the same (or at least, they don't change as often as radio programs do), the radio broadcaster's role is reserved for those objects that constitute the dynamic part of the software components.

Therefore, a new employee who has just joined a company or who hasn't had enough experience with an existing software product can be suitable for developing radio broadcaster objects. It can be a child or a subclass that provides new services for the system.

By now you must have realized that the radio listener's role is assigned to the software components, and that radio listeners are akin to those objects or classes that construct the core components or base framework of the whole software system. These radio listeners are the objects that are most likely to be stable and that require the least number of changes (if any) to the software system. A parent (superclass) or a container object that needs to be extended to provide new functionality is one of those objects.

This analogy, however, is not rigid, because in this design model any object can be a radio listener or a radio broadcaster. Remember that it's always good to plan early and to choose wisely so as not to affect the total development efforts in the later phases.

Keeping this analogy in mind, let's see how we're going to implement this design. In order for radio broadcasters to "transmit" events to radio listeners, they have to implement a custom interface called a RadioBroadcaster interface. Just like any radio listener who wants to "tune in" to the information sent by a radio broadcaster, the listener has to implement a Radio-BroadcastListener interface (see Listing 1).

So, for example, if a class A needs to become a radio broadcaster, class A would implement RadioBroadcaster interface and define its corresponding interface methods (see Listing 2).

The benefit of having a RadioBroadcast object inside RadioBroadcaster is obvious – one doesn't have to redefine all the methods of RadioBroadcast class in all the objects that choose to become radio broadcasters if there are any changes! RadioBroadcast class can be modified (if there's a need to do so) without affecting the radio broadcaster objects as long as the signatures remain the same.

To make life simple, let's assume that class B would like to become a radio listener class and is interested in listening to anything broadcast by class A. So class B implements RadioBroadcastEventListener, as seen in Listing 3.

The two interfaces make use of the following broadcasting channel classes in the process of event communications, i.e., RadioBroadcast and RadioBroadcast-Event – which are mainly extended from java.util.EventObject (see Listing 4).

You may wonder why a class called ServiceBroker suddenly cropped up in the example of class B, acting as a radio listener. For the moment, let's just assume that the class will return an object reference to class A. (The real justifications will become clear later on.)

Sounds simple, right? Yup, but a problem arises. One may ask, What happens if an object needs service or data immediately and without delay? That's a question that specifically involves a term called *nonblocking operation*. In a nonblocking operation (method in this case), data and service have to become available before a method returns or continues processing. As is well known, event propagation takes time. So, the time when service and data will become available (based on the radio broadcast events) is unpredictable.

In solving this problem, I've come up with a way to always make service and data available to a requesting object. An object that provides a service need not be a radio broadcaster or a radio listener, but it's a duty for this object to keep on updating (as neccessary) the type of data required for any interested objects. This data is handled and managed by another intermediate object called a *ServiceBroker*. Once data is made available and is managed by a ServiceBroker, any requesting objects will then retrieve the data directly from the service broker without going through a radio broadcasting channel. This eliminates the process of a source (or, *radio broadcaster*) having to send an event to a listener, and also eliminates the overhead of having to wait or listen for the answering data (or, *feedback*) from the listener.

Do we deviate from our original radio broadcaster/listener analogy here? Not at all. In real life, whenever a radio station wants to get feedback on the popularity of a certain program that's been broadcast to the public, the station will hire an agent or agency to get this information (or, *feedback*) and report on

it. This feedback agency is akin to the ServiceBroker object. Radio stations don't visit your house to get information, but they do hire agencies to get the feedback they need. Thus, based on this analogy, the role of a service broker can't be ruled out. Listing 5 shows an example of ServiceBroker codes.

I suppose at this point you can understand why we used ServiceBroker.getClassA().addRadioBroadcastListeners(this). And there's another benefit to using ServiceBroker. If you take a look at the class B source code that we've previously discussed, you'll find that before listening to an event generated by a radio broadcaster object, there's no need to know what type of object it belongs to! So once again: if in the future the role played by class A, as a radio broadcaster for class B, is replaced by class C, then you won't have to change the codes in class B. This is because when class B calls ServiceBroker.getClassA(), it won't know that a ServiceBroker has returned class C instead of class A – as long as the method signature is the same.

The other issue that should be taken into consideration before using this design model is performance. It's indeed slower, due to event passings relatively compared with direct method invocations. However, I think this design model is much better and worth the effort since it saves a lot of problems with maintaining the codes in later development stages.

One may need to consider which part of the software will use this design model and which part won't, as it can impact the performance if used incautiously. Programmers should realize that with this design model, the messages sent by a radio broadcaster object will have to be unique, otherwise they'll be interpreted by the wrong radio listener! The way I've suggested is to use a naming convention such as packagename.classname.eventname. Not having different sets of event channels for different groups of objects will save programmers a lot of effort.

One last rule of thumb (from my experience) is that you should make the radio listener class an inner class of class A if it's to process the radio broadcasting events generated from within class A (called *intra-events*), and put it outside of class A if it's to handle radio broadcasting events received from outside of class A (called *external events*).

This MVC-style design not only allows programmers to have highly cohesive and lowly coupled objects, but it also allows programmers to decouple the GUI of an application from its data model. Thus the user interface of an application can be thoroughly revamped without the need for any changes to the underlying data model of the application. All swing components are implemented with this design pattern in mind.

In short, you add tremendous value to your company or client by knowing how to design an application, not just build it. Without a methodology covering the details of how you'll partition your application, you're bound to make costly mistakes. Therefore, you should design your system out of small, independent objects. Also, since smaller modules are less complex, new employees can better learn and understand their tasks by being able to focus on the relevant object. And, as they work on more objects over time, they'll be able to develop a broader understanding of the system.

Resource:

Using the JDK 1.1 Delegation Event Model: sunsite.compapp.dcu.ie/IJUG/javaone/sessions/slides/TT21/events-title.html

CODE LISTING

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

About the Author

E Ming Tan was the chief architect of paging and intranet applications at Singapore Telecom. He has worked with Renaissance Software on a JDBC-based Java application, and has worked with LearningByte International – a company that provides customized JFC-based training programs.

futurewave@iname.com

Putting JavaScript Bookmarks to Work

JavaScript can help your bookmarks search the Web with better, faster results

by **Ken Jenks**

Hypertext is wonderful. It allows the Webmaster to link from any page to millions of other computers all over the world. Unfortunately, the Web pages you find will only have the links that were placed by the Webmasters. What if you want more information about a word or a phrase on a page and there's no link?

This brief article shows how to add JavaScript code to your bookmarks – or favorites – thereby allowing you to do some fancy linking where no links exist.

For example, if a Web page contains the word *ennui* and you want a definition for it, you could hunt through your bookmarks for *Webster's Dictionary*, go there, type the word in, go back to your original page, check the spelling of the word, find your way back to the dictionary again and then type it in correctly. Instead, this tiny program allows you to select a word and use a bookmark to search the dictionary for it. A click to your browser's "back" button and you're back again (see Listing 1).

To put this code into a Netscape bookmark, first create a new bookmark in your personal toolbar folder (or anywhere else). Then edit bookmarks and change the properties of your new bookmark. Change the name to "Webster for" and the location (URL) to that JavaScript code, starting with the javascript: tag instead of the usual http:// tag. Now double-click on a word from any document and you can look it up in the dictionary with one click.

This works because of a new feature added to Netscape Navigator 4.0: the ability for JavaScript to detect what text has been selected by the user. When the user uses the mouse to highlight (or select) text in a document, the document.getSelection() method will return a string containing the text.

But there are a few problems with this code. First of all, it doesn't work in Microsoft Internet Explorer (MSIE). MSIE has a different way of detecting current selections. Second, this code won't work if you select text from a framed HTML page.

> ***"You can use these techniques to add a little JavaScript power to every Web site you visit."***

The text you select is in the framed document, not the top document, so this code won't see the selection. Solving these two problems is a bit tricky.

In MSIE 4.0 the document object has a selection property that returns the selection object. You can create a text range object from the selection object by using the createRange() method. Then you can use the text property to get the selected text. See: <http://msdn.microsoft.com/-developer/sdk/inetsdk/help/dhtml/references/objects/obj_document.htm>.

Doing this is incompatible with Netscape's text selection model but that's okay because you'll only be using these bookmarks in one browser anyway. We'll just make two different kinds of bookmarks to cover the two different browsers (see Listing 2).

Note: We could use guard statements to make the same code work in both browsers, but in this case, why bother? You'll only be using the code in one browser anyway.

The next problem – detecting text selections in framed documents – is much harder. We need to walk through all the frames in the parent document and all the frames in each of those frames, recursively, to detect the text selection (see Listing 3).

What a mess! The JavaScript bookmarks all have to be on one line in order to work as bookmarks, but it makes them hard to read. Expanding the code and adding some comments will make it easier to figure out what's going on (see Listing 4).

We use the A and C variables to make it easier to change the JavaScript as the folks at Webster's update their CGI programs, and to standardize the program for other search engines.

See Listing 5 to see how to do the same thing with MSIE.

If you look carefully, you'll notice the MSIE process for selecting text and a slightly different way to index the frames array.

MSIE calls them *favorites* instead of bookmarks. To set this favorite in MSIE, add a new favorite and create it in the links folder. Change the name to "Webster for," then use the right mouse button (or Alt/Enter) to change the properties of the new favorite. Under the Internet shortcut tab, change the target URL to the JavaScript code in Listing 5, including the javascript: tag instead of the usual http:// tag. An error message will appear that you can safely ignore for now. (see Figure 1).

The error message indicates that MSIE doesn't seamlessly support javascript: favorites. There may be further problems getting them to work if MSIE isn't your default browser. To set MSIE as your default browser, view the Internet options, then select the Programs tab. Check the box to tell MSIE to check if it's the default browser, and restart MSIE (see Figure 2).

Note: Of course, this won't work if you want Netscape to be your default browser.

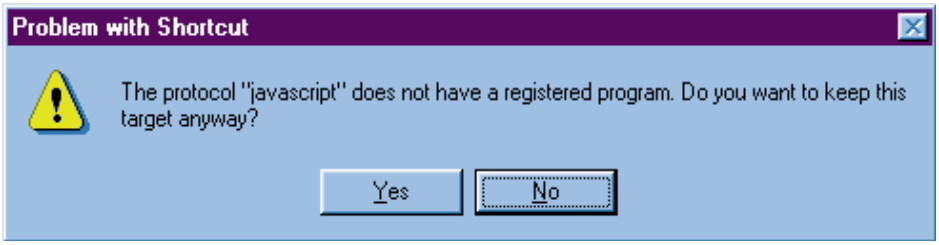


Figure 1: Microsoft Internet Explorer error message when using javascript: favorite

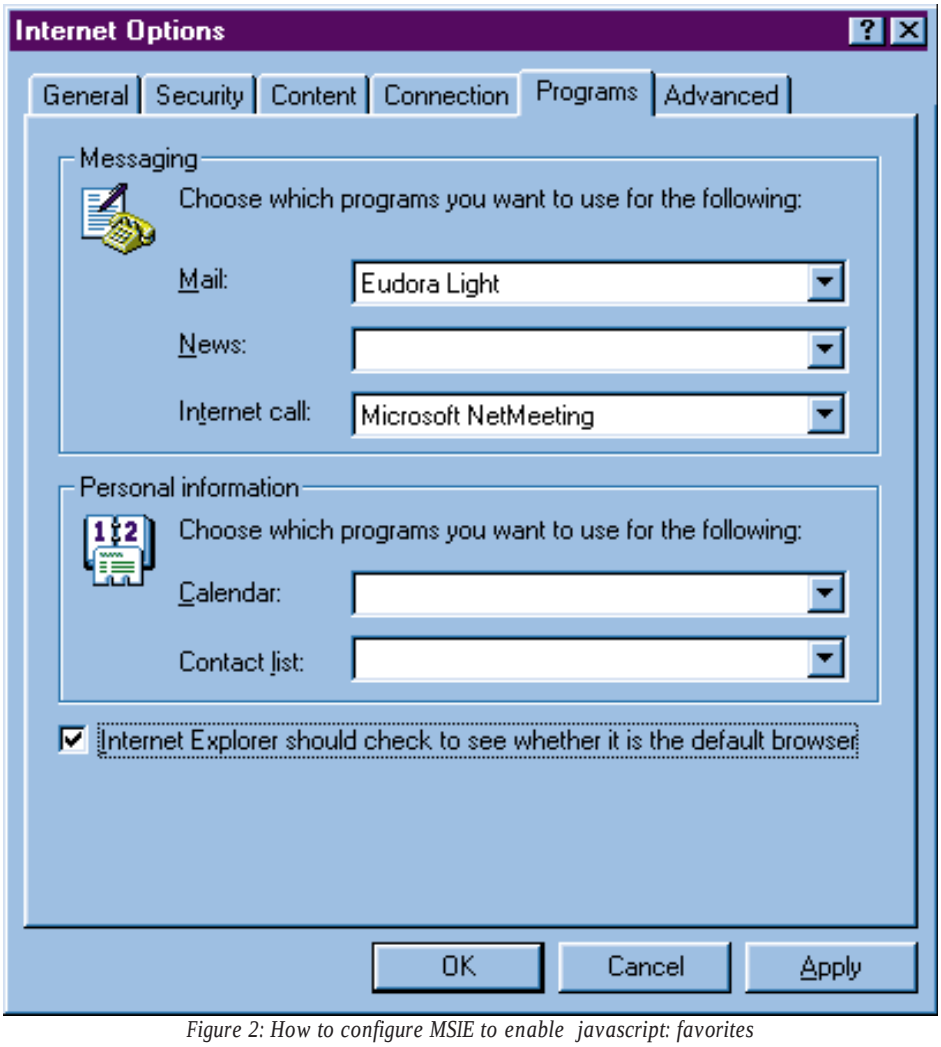


Figure 2: How to configure MSIE to enable javascript: favorites

If Netscape is your default browser you can still use Netscape javascript: bookmarks, but not MSIE javascript: favorites. You'll run into errors if you try to use both. If MSIE is your default browser you can use both with no trouble.

In Listings 3 and 5 we used two variables, A and C, to describe the URL of the CGI program used by the search engine. There are several other options for searching different kinds of engines around the Internet. Use the JavaScript code above and substitute the different values for A and C below:

1. **Searching AltaVista:**
   var A='http://www.altavista.digital.com/cgi-bin/query?pg=q&stq=20&what=web&kl=XX&q=';var C='',
2. **Searching HotBot:**
   var A='http://www.search.hotbot.com/hRes-ult.html?MT=';var C='';
3. **Searching Excite:**
   var A='http://search.excite.com/search.gw?-search=';var C='';
4. **Searching DejaNews:**
   var A='http://www.dejanews.com/dnquery-.xp?QRY=';var C='&defaultOp=AND&svcclass=dncurrent&maxhits=20&ST=QS&format=terse&DBS=2';

How do you figure out the values for A and C in order to harness the power of another search engine? It's a little tricky. You must analyze the HTML form used to start the search, and set up A and C to encode all of the variables, using METHOD=GET.

Let's visit <http://www.lycos.com/>. There's a search form there and it already uses METHOD=GET. Search for a word – say *fiction* – and look at the URL of the results page:

```
<http://www.lycos.com/cgibin/pursuit?match-mode=and&cat=lycos&query=fiction>
```

Now we can construct A and C accordingly.

5. **Searching Lycos:**
   var A='http://www.lycos.com/cgi-bin/pursuit?match-mode=and&cat=lycos&query=';-var C='';

Here's one that doesn't quite fit the pattern. You can select a person's name from the text of a Web page and then use this code to search switchboard.com (see Listings 6 and 7).

Listing 7 is a little different because we need to split the selection into a first and last name, then send those as two different variables.

A Related Technique

Here's another little JavaScript bookmark/favorite that works in both MSIE and Netscape. It searches AltaVista to find external links to the current Web site. You can use this bookmark/favorite to find related sites as well as to gauge the relative popularity of a site (see Listing 8).

Netscape Communicator 4.06 added a "What's Related" feature that uses a central database of related Web sites from Alexa. Listing 9 shows how you can add this capability to other browsers.

Conclusion

You can use these techniques to add a little JavaScript power to every Web site you visit. If you find another use for the techniques you've learned here, drop me a line.

CODE LISTING

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

About the Author

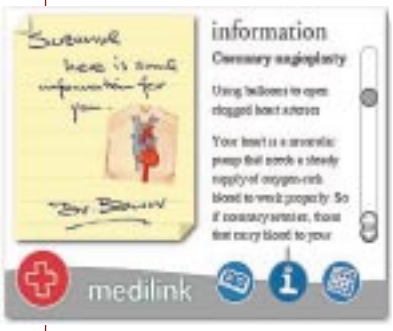
Ken Jenks has been programming for more than 23 years and has been on the Internet since 1984. He holds a BS in computer science and an MS in aerospace engineering, and is working on a Ph.D. in mechanical engineering. In his day job he works for the federal government. Evenings and weekends, he runs a Web-based publishing company called Mind's Eye Fiction (tale.com). He can be reached at MindsEye@tale.com.

MindsEye@tale.com

*The **Java Developer's Journal** billboard at Times Square*

Java Developer's Journal's International Sales Double

(New York, NY) – ***Java Developer's Journal***'s international newsstand sales doubled in 1998, according to the data obtained from the Curtis Circulation Company, worldwide distributor of SYS-CON Publications. ***JDJ*** is making plans to introduce German and French language versions.



Espial Group Releases Kalos Espresso 3.0

(New York, NY) – Espial Group, Inc., released Kalos Espresso 3.0, their third-generation Personal Java GUI toolkit. Kalos Espresso is the industry's only lightweight embedded Java GUI toolkit that develops applications for Java technology-based devices such as PDAs, smart phones, set-top boxes and handheld, mobile and other Internet devices. The corporate mobile and handheld device market is emerging quickly, and Kalos Espresso provides manufacturers a significant time-to-market advantage for developing focused applications that target the various vertical markets including medical, courier, real estate, retail and sales force automation.

For more information call 888 4ESPIAL, e-mail contact@espial-group.com, or visit www.espial-group.com.

KL Group Adds New Heap Analysis Tools to JProbe Profiler 2.0

(Toronto, Ont.) – KL Group Inc., a provider of Java components and advanced development tools, has released a new version of its advanced Java profiling and analysis tool, JProbe Profiler 2.0. The tool that made it easy to search and destroy performance bottlenecks in Java code is now being enhanced by powerful new heap analysis tools that help find and eliminate memory leaks. In addition to comprehensive time and memory profiling with an intuitive call graph interface, developers will be able to use JProbe Profiler 2.0 to profile applications written on either Windows or Solaris using the Java Development Kit 1.1 and 1.2 software. The announcement coincided with Sun's announcement of the availability of JDK 1.2 made during the Java Business Expo.

For more information visit KL Group's Web site at www.klgroup.com.

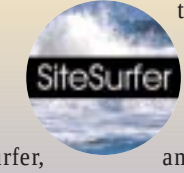
Sun's Java Software Selects Sedona to Showcase New Java JDK 1.2 Technology

(Limerick, PA) – Sedona GeoServices, Inc., a Scangraphics company, announced that it has been selected to join Sun Microsystems, Inc., to showcase the power and performance of Sedona SpatialVision.

SpatialVision provides easy-to-use, timesaving features to optimize a business's rapid return on information investments by exploiting the geospatial components of business data. By displaying geographic information on maps, companies can use SpatialVision to better manage accounts, provide visual decision support information and optimize site planning. SpatialVision enables spatial information to be more easily queried, distributed and viewed through the entire enterprise.

For more information visit their Web site at www.sedonageo.com.

DevTech Announces SiteSurfer 1.0



(Columbia, SC) – DevTech, Inc., has made available their latest product, SiteSurfer, a powerful and easy-to-use searching and navigation software application for Web sites, networks and local files. SiteSurfer allows Web site administrators to provide visitors to their site with powerful searching and navigation capabilities independent of the facilities provided by their Web-hosting service. SiteSurfer also supports searching content on hard drives and local area networks, with built-in support for text, HTML, Ami Pro, Windows Write, Microsoft Word, Microsoft Word for Windows and WordPerfect files.

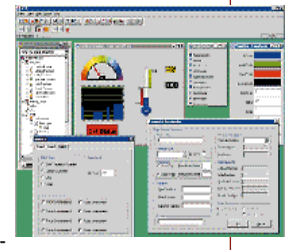
For more information call 803 790-0923 or visit www.devtech.com.

Rapid Logic Announces 2.2 Release of JavaControl

(Alameda, CA) – Rapid Logic Inc., has released JavaControl 2.2, a software development kit that provides SNMP support, increased power and reduced development time for Java-based networked device management.

JavaControl

Version 2.2 of JavaControl unleashes the full power of Java for managing a device's fundamental data elements without requiring an embedded Java Virtual Machine, expensive dedicated consoles or any Java programming expertise. Companies embracing JavaControl include leaders in the networking, datacom and telecom industries.



For more information contact the company at www.rapidlogic.com.

Cloudscape Ships New Generation of Embeddable Java Database

(Oakland, CA) – Cloudscape, Inc., has enhanced the industry's first embeddable Java database designed for distributed, off-line and mobile computing. The new versions of Cloudscape are designed to simplify the deployment and management of mobilized applications with two significant technology innovations, including VTI for data integration and LUCID for sophisticated application synchronization. More than 200 companies now rely on Cloudscape for distributed, mobile and "occasionally connected" computing solutions.

For more information visit their Web site at www.cloudscape.com.

Object Management Group

www.omg.org

NetBeans Announces Support for the Java Development Kit 1.2

(New York, NY) – NetBeans has announced that its Java IDE, NetBeans DeveloperX2, supports and runs on Sun Microsystems' Java Development Kit. This latest release of the JDK provides a rich feature set of new class libraries and tools, making it easier than ever for developers to create portable, distributed, enterprise-class applications. Sun's announcement of the availability of the next version of the JDK was made during the Java Business Expo in New York.

The final release of NetBeans DeveloperX2 2.1 is available this month. A concurrent version supporting JDK 1.1.x will also be available. A full release of this edition of the IDE is due this spring.

For more information visit their Web site at www.netbeans.com.

Telecom Italia Deploys Jacada from CST

(Atlanta, GA) – CST announced that Telecom Italia, the principal provider of domestic and international telecommunication services in Italy, has licensed Jacada from CST to deploy Java graphical client access for its mainframe applications. The initial project improves the ability to track how telephone calls are routed throughout Telecom Italia's national telecommunications network, consisting of more than 300 local networks connected by a private IP backbone.

You can visit CST's Web site at www.cst.com.

Secant Technologies Announces Secant Extreme Enterprise Server

(New York, NY) – Secant Technologies, a provider of advanced software technology for building multitier applications, has announced the availability of Secant Extreme Enterprise Server for Enterprise JavaBeans.

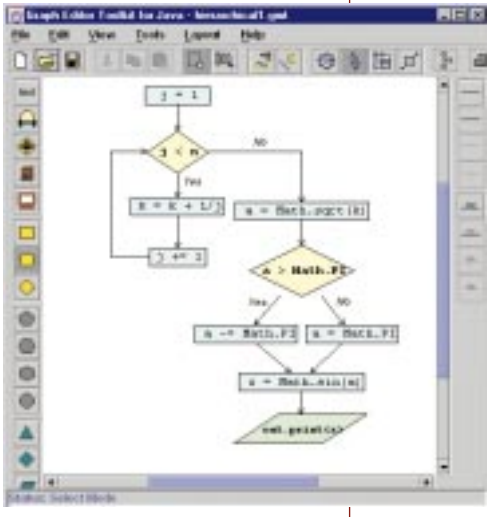
Secant Extreme Enterprise Server for EJB provides a complete environment for assembling, deploying and maintaining scalable, multitier business systems. The product is based on Secant's time-tested and proven Object Transaction Monitor and object data management technology. This technology includes superior implementations of CORBA ORBs and services such as transactions, security, persistence, events, concurrency and locking.

For more information visit their Web site at www.secant.com.

Network Design System Powered by Graph Layout

(Berkeley, CA) – Tom Sawyer Software, a vendor of graph layout technology, has incorporated its Graph Layout Toolkit within NetSuite Advanced Professional Design, a network design system that automates network design, validation and documentation. The Graph Layout Toolkit provides scalable network layout components for visualizing large multilevel network designs. NetSuite Advanced Professional Design helps network professionals create and maintain a living model of their network, thereby enabling them to evolve their networks to meet growing business demands accurately, promptly and efficiently.

For more information visit www.tomsawyer.com.



Data Representations Releases Simplicity for Java 1.1

(New York, NY) – Data Representations, Inc., is launching version 1.1 of Simplicity for Java. It's written completely in Java 1.1 and runs on any Java-enabled platform, including Linux, OS/2, Solaris, AIX, HP-UX, Windows 95/98/NT and others.

By using the Simplicity for Java IDE, developers build through a visual mode that is instantly updated to reflect any changes made to the program's source code. This reduces development time, promotes programming accuracy and challenges the need for the traditional three-step development practice of code, compile and test.

For more information visit their Web site at www.datarepresentations.com.

BEA WebLogic Promotes New Vice President of Marketing

(San Jose, CA) – BEA WebLogic's Katherine Barnhisel has been named BEA's vice president of marketing. Barnhisel visited the ***JDJ*** booth during the recent Java Business Expo in New York.

Katherine Barnhisel visits the ***JDJ*** *booth at the Java Business Expo*

RasterMaster for Java Version 2.0 Released

(Newton, MA) – Snowbound Software has announced the release of version 2.0 of its RasterMaster for Java-imaging components. Based on its RasterMaster technology, the newest Java version offers more capability and functionality than before.

One of the new features available with RasterMaster for Java version 2.0 is its capability to print high-resolution images from within Java. RasterMaster for Java aids the language's promise of "write once, run anywhere," as the printing dilemma has now been solved by Snowbound's research and development team.

For more information call 617 630-9495, e-mail salesp@snowbnd.com, or visit their Web site at www.snowbnd.com.

THE MARKETPLACE

ABLE SOLUTIONS

Enter the realm of browsable store building and administration – from your browser. Build "your_site.com" with secure Merchant Credit Card Processing. Maintain inventory, add discounts and specials to keep your customers coming back. Increase sales with cross selling and membership pricing.

You can reach Able Solutions at www.ablecommerce.com or at www.ablesolutions.com.

11700 NE 95th Street, Suite 100, Vancouver, WA
360 253-4142

ALLAIRE CORPORATION

At Allaire, our focus is to empower developers with the tools and knowledge to deliver on the promise of the Web as a platform for crucial business applications. We offer a wide range of flexible programs, including professional education, consulting, technical support and partner programs that complement our existing documentation and online developer's center.

You can contact Allaire at www.allaire.com.

One Alewife Center, Cambridge, MA 02140
888 939-2545

CATOUZER INC.

With Synergy 1.0 Web application framework, creating custom Intranet applications is a breeze. The Synergy Application Development Kit (ADK) gives you the tools to rapidly develop your custom apps, which can be fully integrated and managed under the Application Services Layer (ASL).

For more information contact Catouzer at www.catouzer.com.

1228 Hamilton Street, Suite 501, Vancouver, B.C. V6B 2S8, Canada
604 662-7551

EPRISE CORPORATION

If your customers are looking for a content management solution, Eprise Participant Server can save you time and resources. Participant server is a flexible content management framework that enhances high-value business relationships through the delivery of timely, targeted, Web-based communications. Get in touch with us today (www.eprise.com) to learn more about the Eprise Participant Server FastStart Kit for Allaire ColdFusion Developers.

1671 Worcester Road, Framingham, MA 01701
800 274-2814

THE IGNEOUS GROUP, INC

The Igneous Group provides custom solutions for e-commerce business information systems and dynamic content publishing. We build systems to your specifications and integrate them with your existing backoffice data and business infrastructure. We can help you get your site online, or identify how to make it function more efficiently.

For more information, check out The Igneous Group at www.igneous.com.

541 Seabright Avenue, Santa Cruz, CA 95062
877 469-ROCK

INTERMEDIA, INC.

Our advanced virtual hosting packages (powered by Microsoft Windows NT and Internet Information Server 4.0) offer an environment supporting everything today's advanced Web developer or sophisticated client could ask for. Complete ODBC support is available on plans B and C. We support Microsoft Index Server on all hosting plans.

Contact Intermedia, Inc., at www.intermedia.net.

953 Industrial Avenue, Suite 121, Palo Alto, CA 94303
650 424-9935

LIVE SOFTWARE

The power of Java and the simplicity of ColdFusion, <CF_Anywhere> gives ColdFusion programmers the ability to leverage all the power and flexibility of Java using the familiar ColdFusion Markup Language (CFML). <CF_Anywhere> was perfected for CF developers, CF administrators, ISPs and everyone.

For more information about <CF_Anywhere>, go to www.cfanywhere.com.

5703 Oberlin Drive, Suite 208, San Diego, CA 92121
408 996-0300

VIRTUALSCAPE

Why host with Virtualscape? Nobody else on the Internet understands what it takes to host ColdFusion like we do. From Fortune 500 extranets to e-commerce sites and more, developers recognize our speed, stability, reliability and technical support.

Virtualscape can be reached at www.virtualscape.com.

215 Park Avenue South, Suite 1905, New York, NY 10003
212 460-8406

THE GRIND

by **Java George**

Java – Into Its 4th Year

In the final months of 1998 it was worth reflecting on the networked application platform and its star player, Java, as we headed into the new year. Many of you have written and put forward your own assessment of the Java movement as it rode into 1999, representing the fourth year of Java as a commercially available development platform.

This is the first installment of our theme "Java – Into Its 4th Year," and we will review the platform independence promise of the Java platform: run anywhere, anytime. It's not an all or nothing proposition.

Never has more hype surrounded a basic notion of platform independence as Java's "run anywhere, anytime" promise. This was particularly acute with Java on the client side (in browsers). Unfortunately, Sun's marketing messages were taken completely out of context and Java developers assumed that any piece of compiled Java code would magically run not only on all virtual machines and platforms, but in all circumstances as well. Why would any experienced developer fall into such a predicament?

The marketing message should probably have been "Java, run anywhere, anytime – with the appropriate amount of effort!" It's certainly possible to create a Java client that runs across the vast majority of possible client configurations. Creating a Java client that runs across all possible client configurations is not significant, that is, it's not necessary to cover all permutations to achieve the desired effect. In other words, if out of 100,000 end users 10 can't access your Java client because they're running OS/2 and the Navigator 2.0 browser, does that constitute a failure in deployment? I should say not.

A well-engineered Java client is extremely successful against a wide cross-section of possible client configurations easily covering 90% of possible end users. Witness the Yahoo! game site and its multiplayer Java games: chess, backgammon and checkers. The Yahoo! games constituency runs across Macintosh, Windows and Solaris Workstations, and various versions of browsers across each. Witness the PROGRESS Apptivity 3.0 Java client that supports Netscape and the IE 3.x and 4.x browsers across all major OS platforms. "Support" doesn't translate into "works every time without any effort on your part"; it translates into "will be portable with a reasonable amount of effort by the developer and the vendor."

On the server side, Java platform independence has received additional scrutiny. While it's clear that the UI layer presented the toughest platform challenges, the server side is not immune from slight discrepancies in the JVM. The most important thing to remember is that server-side processes, particularly those oriented at high-transaction business functions, are very reliant on the robustness of the JVM implementation. A server process with 1,200 threads and 1,500 object instantiations is no angel; it's significantly dependent on its Java container. Such a beast running on JVM 1.0.2 would most certainly come to a crashing halt, while under JVM 1.2 it would most likely hold its own. It would perform even better if such a complex process was abstracted on top of a competent CORBA infrastructure.

Once again, this shows the importance of layering the Java solution and insulating the developer from the challenges faced in achieving platform independence and performance. On the client side, the developer using a proven, supported Java client architecture will achieve success far and above a counterpart creating the client from scratch. At the very least, the Java client should take advantage of a foundation class such as JFC/Swing, AFC or Netscape's IFC. On the server side, the developer building on top of JDK 1.2, Corba and EJB will get to successful deployment and scalability while the developer building from scratch will most likely never achieve deployment at any reasonable scale!

Java is heading into its fourth full year as a legitimate language and platform, and with it, developers can achieve platform independence with a reasonable effort. How much effort will be required to port an ActiveX control across Unix, Windows and Mac platforms? How much effort will be required to port a significant C or C++ executable across these platforms?

Platform independence is not an all or nothing proposition. It's a question of practicality, and a question of reasonable effort leading to significant value!

"A well-engineered Java client is extremely successful against a wide cross-section of possible client configurations..."

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@apptivity.com