JDJEDGE CONFERENCE & EXPO OCTOBER 1–3, 2002 SAN JOSE, CA

JAVA™ DEVELOPER'S JOURNAL

The World's Leading Java Resource

September 2002 Volume:7 Issue:9

JAVADEVELOPERSJOURNAL.COM

FULL CONFERENCE PROGRAM

WEB SERVICES EDGE

INTERNATIONAL WEB SERVICES CONFERENCE & EXPO

McENERY CONVENTION CENTER, SAN JOSE, CA

INTEGRATING APPLICATIONS ▶▶▶ CONNECTING ENTERPRISES

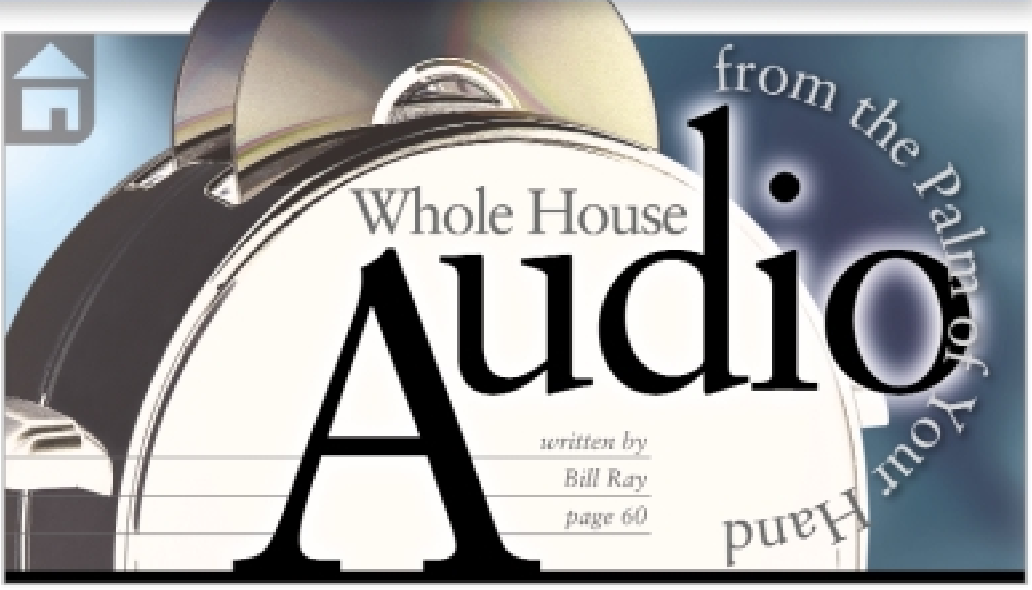
CONFERENCE: October 1-3, 2002

EXPO: October 2-3, 2002

CALL TODAY!

SAVE $200

▶▶▶ INSIDE PAGE 75



RETAILERS PLEASE DISPLAY UNTIL NOVEMBER 30, 2002

$5.99US $6.99CAN

0 09281 01751 6 10>

SYS-CON MEDIA

sonic

www.sonic.com

zerog

www.zerog.com

apple

www.apple.com

bea

www.bea.com

JAVA DEVELOPER'S JOURNAL

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

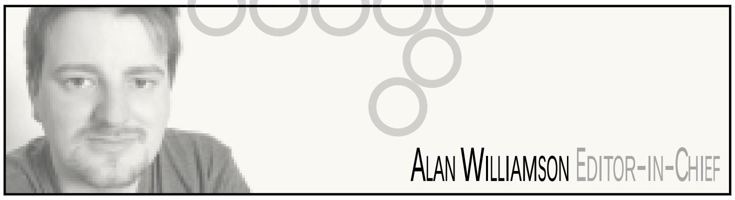
EDITORIAL
EDITOR-IN-CHIEF: ALAN WILLIAMSON
EDITORIAL DIRECTOR: JEREMY GEELAN
EXECUTIVE EDITOR: NANCY VALENTINE
J2EE EDITOR: AJIT SAGAR
J2ME EDITOR: JASON R. BRIGGS
J2SE EDITOR: JASON BELL
PRODUCT REVIEW EDITOR: JIM MILBERY
FOUNDING EDITOR: SEAN RHODY

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

WRITERS IN THIS ISSUE
BILL BALOGLU, JACQUIE BARKER, MICHAEL BARLOTTA, JASON BELL, JASON R. BRIGGS, JOHN CHAMBERLAIN, DUANE FIELDS, KEDAR GODSE, BRUCE HOPKINS, BILLY PALMIERI, BILL RAY, SAAD REHMANI, BRIAN A. RUSSELL, AJIT SAGAR, KUNAL SHAH, ALAN WILLIAMSON, BLAIR WYMAN

SUBSCRIPTIONS:
FOR SUBSCRIPTIONS AND REQUESTS FOR BULK ORDERS, PLEASE SEND YOUR LETTERS TO SUBSCRIPTION DEPARTMENT
SUBSCRIPTION HOTLINE: SUBSCRIBE@SYS-CON.COM
COVER PRICE: $5.99/ISSUE
DOMESTIC: $49.99/YR. (12 ISSUES)
CANADA/MEXICO: $79.99/YR. OVERSEAS: $99.99/YR.
(U.S. BANKS OR MONEY ORDERS). BACK ISSUES: $10/EA., INTERNATIONAL $15/EA.

EDITORIAL OFFICES:
SYS-CON MEDIA 135 CHESTNUT RIDGE RD., MONTVALE, NJ 07645
TELEPHONE: 201 802-3000 FAX: 201 782-9600
JAVA DEVELOPER'S JOURNAL (ISSN#1087-6944) is published monthly (12 times a year) for $49.99 by SYS-CON Publications, Inc., 135 Chestnut Ridge Road, Montvale, NJ 07645. Periodicals postage rates are paid at Montvale, NJ 07645 and additional mailing offices. POSTMASTER: Send address changes to: JAVA DEVELOPER'S JOURNAL, SYS-CON Publications, Inc., 135 Chestnut Ridge Road, Montvale, NJ 07645.


ALAN WILLIAMSON EDITOR-IN-CHIEF

'Tale of Two Camps'

It has been far from quiet on the ***JDJ*** forums front this month – a result of the recent news item we broke regarding the now infamous Gartner report. This report claimed that there will be a major shortage of Java developers in the forthcoming year. Which is good news…I think! So why are you all in an uproar?

Opinions appear to be divided on this news, with two camps emerging: the Java developers who are currently out of work and struggling to find Java contracts versus the employers who claim they can't find qualified Java people. Tales of mock job listings (on- and offline) with phantom positions are the stuff of legends. But is there some fire to all this smoke? We all have our own agency horror stories. For my sins, I was once offered the post I was trying to recruit for! Which camp has it right? Who can really tell? Only time will.

Looking through the posts and other online sites such as Slashdot, it's clear that as a community drawn together by a common language, we may have suffered from the early hype of Sun and others. The hype surrounding Java gained fever pitch only a couple of years after Java appeared in Web browsers in its legendary gray rectangle.

The next wave hit when the Java Servlet API proved Java was a serious contender on the server side, an area largely untouched by platform-independent solutions. It can be argued that had it not been for the Servlet API, Java would not be considered the powerhouse it is now. It was the Servlet API that firmly put the power of Java on the server side at a time when we were asking more from our Web sites, looking for innovative ways to produce dynamic content – the height of the dot.com fever pitch.

With this hype came the usual "jump-on-the-bandwagon" brigade; universities churning out so-called Java programmers, companies guaranteeing (and some still are!) Java certification for a given fee, and agencies promising all the gold in Fort Knox for salaries have all contributed toward a watering-down of the general Java skill base.

In this drive to get people into the "in-crowd" we seem to have lost the core competency that should bind us together: software engineering, not Java. From that perspective it is easy to sympathize with the employer who is desperately looking for skilled software engineers (AKA Java developers) and not the "…in 21 days" adopters.

What of the other camp, which claims the jobs aren't there?

That may be, but were the jobs/positions there in the first place? The dot.com boom managed to artificially inflate everything, particularly the recruitment market. Java's popularity was at its peak during this period, and you need only track the exhibitor lists over all the past JavaOnes to see this trend play out. There has not been a computing language that has caught the imagination of the world's media like Java has, and I believe we are feeling the backlash of this early, misdirected hype. We got caught up in selling Java the technology and forgot what the tool really is: a programming language to solve problems.

At the end of the day we are software engineers, designed to solve problems. That is what we are trained, paid, and get out of bed for. That we choose Java to express our solutions is a bonus, and as Jason Briggs commented this month in his editorial, we have many feathers in our cap and strings in our bow, but Java is the one we definitely prefer.

Java's power is in its sheer beauty. The ability to write a single piece of code and have it run in a plethora of devices, from high-end enterprise machines to handheld devices and mobile phones, is the result of the engineering genius that lies underneath the covers for us all to utilize.

Forget the razzamatazz and the glitz of the dot.com era; we have real work to do, real-world solutions to deliver with a tool that can save us time and energy.

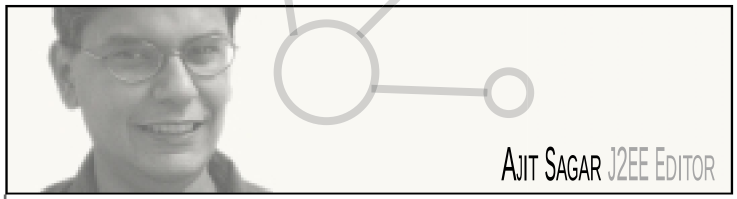
Java isn't .NET…it's .NOW!

alan@sys-con.com

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

motorola

www.motorola.com



AJIT SAGAR J2EE EDITOR

Verifying Java for the Enterprise

About three months ago, my two-year old son discovered the word "cup." He would call everything a cup, though he had no clue what a cup was. Finally we figured out a way for him to call a cup a cup – we pointed to a cup every time he uttered the word. In my technological world of J2EE, I could map this activity to certification and verification.

Today the market for J2EE is mature enough that application and framework alternatives are available from several competing sources. As a result, there's a need for standard metrics and credentials that can be used by companies to evaluate short-list vendor solutions for their specific requirements. From the solution providers' perspective, credentials help them get their foot into a prospective client's door. From a technical perspective, these credentials can take the following forms:

- The technical team's profile
- The maturity and reliability of their product suite in the context of the client's environment

Given the existing and growing base of J2EE architects and developers in the market, the personal profile of each team member plays a large role. This is where certification comes in. Various levels of Java certification that address different facets of J2EE are offered by Sun Microsystems, as well as other major Java proponents like IBM and Oracle. At the same time, major application server vendors such as BEA and IBM offer their own certification that verifies the ability of a person to use J2EE in their app server–specific environments.

When building a J2EE-based solution, a major concern of a solution provider is the viability of the tools and development frameworks used. And one of the biggest messages of the Java platform is that it's portable. Sun helped bring the message together with their J2EE Blueprints and Pet Store reference application.

The basic premise of J2EE portability is the ability of an application to function across multiple application servers, which is where verification comes in. The good news is that the recently released Java Verification Program from Sun Microsystems addresses this issue. It's designed to identify enterprise applications developed with J2EE technology and intended to be portable across different J2EE implementations. The program outlines the tests needed to receive the Java verification certification. Products that complete the Java AVK for the Enterprise testing process can apply for the Java Verification Program and Trademark license. More information on the Java Verification Program can be found at http://java.sun.com/j2ee/verified/index.html.

Several app-server vendors offer add-ons that may not be portable across other app servers, but provide a tremendous value-add.

For example, Macromedia has developed their own version of the Pet Store called the Pet Market – touted as a "rich Internet application" – built on Macromedia's MX family of products. According to Macromedia, the Pet Market serves as a blueprint for best practices for usability, architecture, and coding. The Pet Market serves as a reference application for portability across a J2EE application server for server-side components, and portability across commerce platforms for Web applications. I call this a combination of good technology, good sense, and good marketing.

As their initiatives mature, there's a growing need for good reference sources in the market to guide developers through the design process.

• • •

I would like to mention a couple of good books I picked up from a new publisher – Apress. *Java Collections* is an excellent reference with clear and concise examples. And *Java FrontEnd Technologies* had some very useful guidelines for JSP and servlet design in the context of J2EE.

ajit@sys-con.com

AUTHOR BIO
Ajit Sagar is the J2EE editor of JDJ and the founding editor of XML-Journal. Ajit is the director of engineering with Controlling Factor, a leading B2B software solutions firm based in Dallas, and is well versed in Java, Web services, and XML technologies.

Primary Keys and Container Managed Persistence in EJB 2.0

Single versus compound primary keys

WRITTEN BY
SAAD REHMANI

This series of articles will walk you through the details and some of the decisions that must be made when implementing container-managed persistence in Enterprise JavaBeans.

Of course, there is the usual discriminator. These articles are not based primarily on the EJB specification and what you can and cannot do with EJBs; instead, they concentrate on information derived from hard-earned experience you'll find useful when dealing with EJBs.

What's a Primary Key?

The primary key in an EJB is the subset of its attributes, which are guaranteed to be unique. Informing the container of the contents of an entity's primary key allows it to store and later, using the PK, retrieve the same entity. Primary keys always provide a handle to the entity, regardless of whether it's in memory or storage.

There, that sounded abstract enough, on to reality. Persistence mechanisms in EJB containers, at least those that are efficient and widely accepted, are closely tied to databases. Furthermore, although there are a few object-oriented databases in the market, their acceptance is limited compared to their relational ancestors.

In essence, relational databases manage reads, writes, and searches on tables that are made up of columns and rows. Entity beans map cleanly to tables; each column maps to an attribute; each row maps to an entity. This is not true in coarse-grained approaches where one entity may be responsible for multiple rows in multiple tables, but that approach is no longer a necessity due to performance gains made in EJB 2.0's handling of a finely grained object model.

What's a Good Primary Key?

Choosing which unique part of an entity's attributes the primary key should be composed of is not an easy task. It gets exponentially harder to guarantee amid changing requirements. For example, the first and last name attributes in an entity modeling Employees might be considered unique at design time, but this might not hold true in the long run. Primary keys that are subsets of their attributes are troublesome because uniqueness tends to fade as data accumulates and new attributes are added to the entity.

At times, adding new attributes can mean adding a new differentiator, which must be factored into the logical primary key of the entity. As a result, earlier guarantees of uniqueness are no longer valid. If we were to add a middle initial attribute to the example entity we used earlier, it would have to be added to the primary key, resulting in a lot of refactoring. Figure 1 shows the difference adding an attribute to an entity can have on both a single and compound primary key.

Although it's possible to use a string instead of an integer, this approach has several problems, such as slightly slower performance when doing lookups based on strings, string concatenation not being an effective way to extend the primary key, and the fact that containers support only autoincrementing integral primary keys.

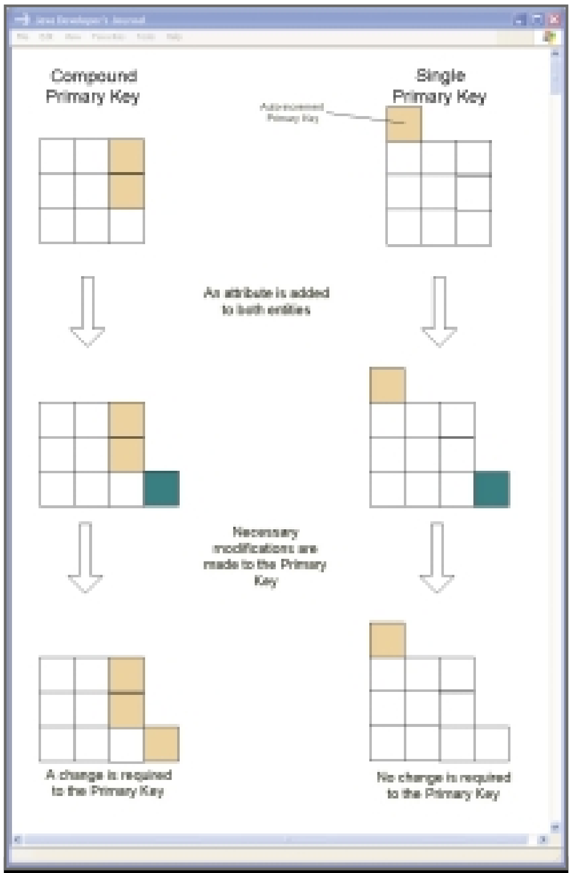

Java Developer's Journal
Compound Primary Key
Single Primary Key
Auto-increment Primary Key
An attribute is added to both entities
Necessary modifications are made to the Primary Key
A change is required to the Primary Key
No change is required to the Primary Key

FIGURE 1 Adding an attribute

rational

www.rational.com

AUTHOR BIO
Saad Rehmani is senior software engineer at a small startup that does big things. His current responsibilities include extensive work with J2EE in general and EJB 2.0 in particular. Before realizing how awesome Java was, Saad was heavily involved in various projects ranging from kernel modules to pseudo-realtime state propagation between clusters.

Another issue with multiattribute primary keys arises when working with some container-managed relationships. When dealing with a many-to-many relationship between two entities, the underlying database table that's modeling this relationship consists of columns that match the primary keys from both entities. If the primary keys of both entities are compound and a common attribute name is shared between them, the database layer cannot differentiate between them at the column level.

Because these hard lessons have been learned multiple times over, using an automatically generated integer is something I would highly recommend. Since it's autoincremented by the container as the primary key for an entity, not only is it the easiest to implement, it's also the most flexible over time. The only caveat to using automatically generated integral primary keys is the container cannot enforce uniqueness. If we were to create three different entities with the same attributes and used an autoincremented integer as the primary key, the container would not complain about duplication since the autoincremented integer primary key would still be unique. In some cases, this may be valid and duplicates of the logical primary key might be supported by the business logic, while other scenarios might not allow duplication.

Enforcing Logical Primary Keys in the Database

One way to avoid this pitfall is to add constraints to the database that don't allow this to happen. Even though it makes the existence of a database underneath the persistence layer visible, ruining encapsulation, it leverages what databases do much better than EJB containers: it keeps track of data. A quick detour through database constraints from an EJB perspective might be helpful.

Although most EJB containers are able to generate the underlying persistence schema, very few people use it directly in production, mostly because the tables created by the container contain no constraints. Not only are constraints important in terms of disallowing bad data, they also provide important performance hints to the database. All relational databases are able to define the columns in a table that make up the primary key. They also enable us to define unique indexes. In case you're wondering, primary keys are specialized unique indexes.

If logical uniqueness is not enforced on the entity layer via a multiattribute primary key, enforcing it on the database layer is a useful and effective method of guaranteeing uniqueness. Since both the logical primary key and the autoincrement attribute need to be uniquely independent of each other, defining the primary key on the database level to be logical and defining a unique index for the actual autoincremented primary key achieves a constraint that disallows duplicate entries, and an index that allows for fast lookups when searching by the actual primary key.

The programmatic alternative is to create an entity finder based on the logical primary key, then check for the nonexistence of an entity every time before creating it, thus guaranteeing that no duplicates are generated.

Summary

In an evolving marketplace, business requirements keep changing. As promises of matchlessness weaken to assurances and less, and modeled entities take on more and more properties, something as important as the primary key of an entity should be as constant as possible. This can best be achieved by a single primary key that's autogenerated, an integer, and not dependent on the portion of the entity that's sure to change over time.

bonga@aitchisonians.org

int

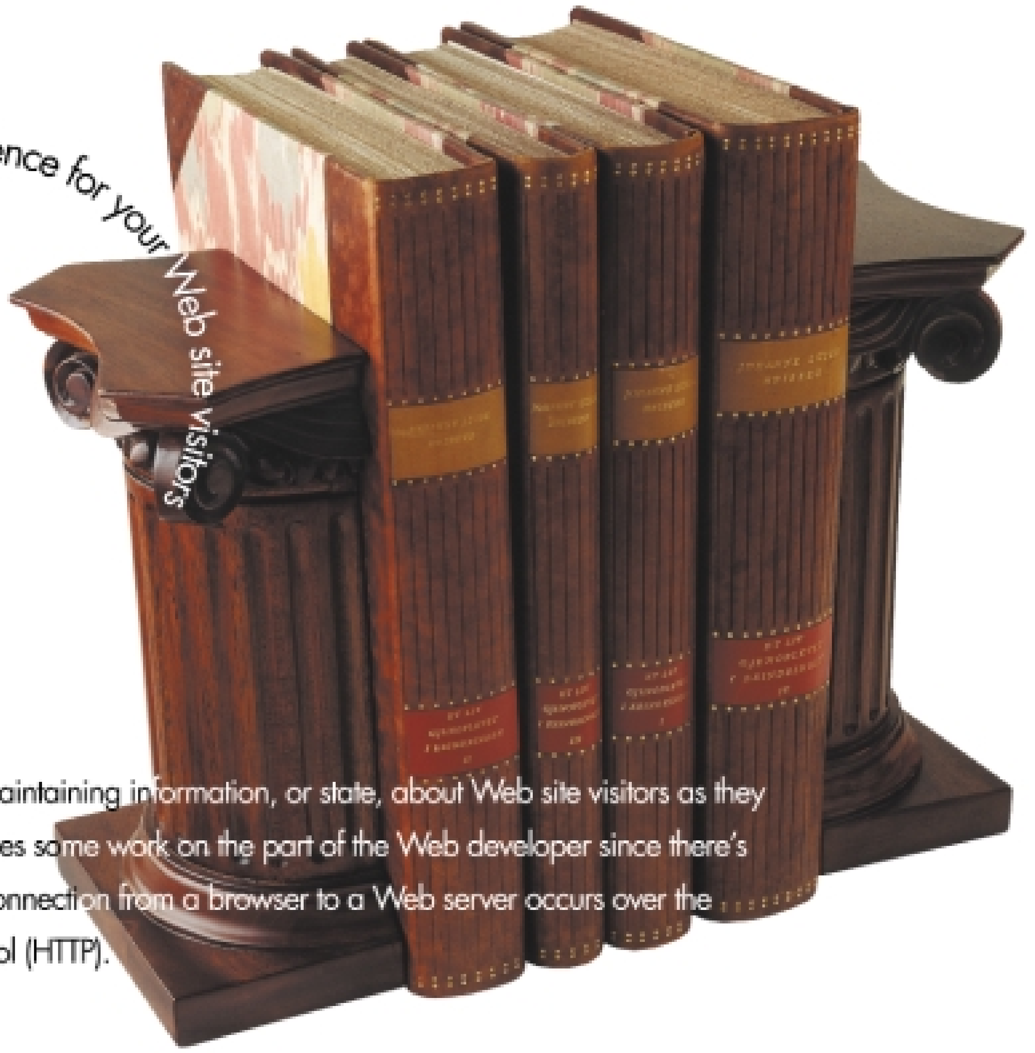
www.int.com

crystal decisions

crystaldecisions.com

infragistics

www.infragistics.com



Create a better experience for your Web site visitors

written by Brian A. Russell

Session tracking is the process of maintaining information, or state, about Web site visitors as they move from page to page. It requires some work on the part of the Web developer since there's no built-in mechanism for it. The connection from a browser to a Web server occurs over the stateless Hypertext Transfer Protocol (HTTP).

Introduction to Session Management

There are a number of ways to handle session tracking, but our focus is on the easy-to-use yet powerful HttpSession interface provided by the Java Servlet specification. Before we get into the HttpSession interface, let's look at some other ways of maintaining state.

Session-Tracking Techniques

At one time Web developers used Web site visitors' IP addresses to track the sessions. This approach was inflexible and had many flaws. The main problem was that proxy servers eliminated the use of individual IP addresses. Users no longer had unique addresses, so this technique couldn't work properly. Another way of handling session tracking is the use of the HTML hidden field:

```
<INPUT TYPE="hidden" NAME="user"VALUE="Jennifer">
```

This technique required server-side scripting that would dynamically generate the HTML code that contained the "user" field. Server-side code was also required to read the field and match it to information about this user on the server.

Another session-tracking technique is URL rewriting. In this approach, identification field(s) are appended to the end of each URL for a Web site. The following HTML code demonstrates this method:

```
<A HREF="/orderform.htm?user=Jennifer">Order Now!</A>
```

This approach is similar to hidden fields. The difference is that hidden fields can only be used in a form.

A common way of session tracking is the use of cookies. A cookie is information that's stored as a name/value pair and transmitted from the server to the browser. Cookies containing unique user information can be used to tie specific visitors to information about them on the server. The Java Servlet specification provides a simple cookie API that allows you to write and retrieve cookies. The complete API can be found on Sun's Java Web site, http://java.sun.com. The following code shows how to create a new cookie:

```
Cookie user = new Cookie("user","Jennifer");
user.setMaxAge(3600);
response.addCookie(user);
```

oracle

www.oracle.com

This code creates a cookie with a name of “user” and a value of “Jennifer”. The cookie’s expiration date is set with the setMaxAge() method to 3,600 seconds from the time the browser receives the cookie. The following code demonstrates how you would retrieve the value for a specific cookie:

```
String user = “”;
Cookie[] cookies = request.getCookies();
if (cookies != null) {
 for (int i = 0; i < cookies.length; i++) {
  if (cookies[i].getName().equals(“user”)) user =
    cookies[i].getValue();
 }
}
```

In this code, an array of cookies is retrieved from the HttpServletRequest object using the getCookies() method. The array is walked through until a cookie with the name of “user” is returned by the getName() method. Once the cookie is found, the getValue() method is called to retrieve the value of the cookie.

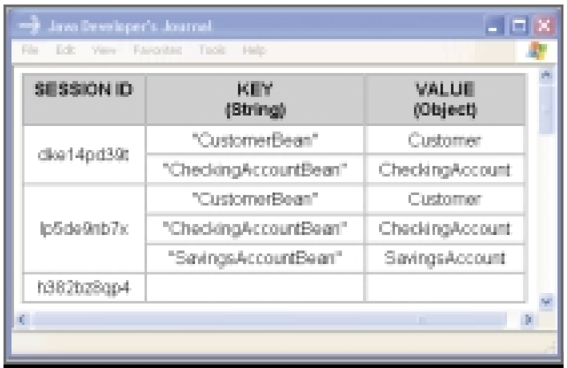
The use of cookies provides a flexible and easy option for handling session tracking; however, it does present some problems. The information in the cookie is stored on the client’s browser in a text file that can be easily read and manipulated, and this information is transmitted unsecured across the Internet. But the main problem is that they can be disabled through a setting in the Web browser. Web sites that rely on cookies for session tracking will be unable to track users who have disabled cookies.

Sessions in Java

The session management techniques we have looked at so far all have a common security issue: they transmit data in plain text. A powerful session-tracking solution is needed that’s more secure and flexible. This is where the Java HttpSession API comes in. The HttpSession API provides a simple mechanism for storing information about individual users on the application server. The API provides access to a session object that can be used to store other objects. The ability to tie objects to a particular user is important when working in an object-oriented environment. It allows you to quickly and efficiently save and retrieve JavaBeans that you may be using to identify your site’s visitors, to hold product information for display on your online store, or to track products that potential customers have placed in their shopping carts.

A session object is created on the application server, usually in a Java servlet or a JavaServer Page. The object is stored on the application server and a unique identifier called a session ID is assigned to it. The session object and session ID are handled by a session manager on the application server. Figure 1 illustrates this relationship. Each session ID assigned by the application server has zero or more key/value pairs tied to it. The values are objects that you place in the session. Assign each of those objects a name, and each name must have an object with it because a null is not allowed.

For this session-tracking technique to work, the session ID must be sent to the client’s computer. A cookie is used to store the session ID on the Web site visitor’s computer. This is automatically handled by the application server. Simply create the session object and begin using it. The application server will, by default, create the session ID and store it in a cookie. The browser will send the cookie back to the server every time a page is requested. The application server, via the server’s session manager, will match the session ID from the cookie to a session object. The session object is then placed in the HttpServletRequest object and you retrieve it with the getSession() method.



SESSION ID	KEY (String)	VALUE (Object)
dke14pd39t	"CustomerBean"	Customer
	"CheckingAccountBean"	CheckingAccount
lp5de9nb7x	"CustomerBean"	Customer
	"CheckingAccountBean"	CheckingAccount
	"SavingsAccountBean"	SavingsAccount
h382bz8qp4		

FIGURE 1 Session IDs and session objects

As we discussed earlier, some Web site visitors will have cookies disabled in their browsers. To get around this problem and continue using sessions, use URL rewriting in your code. URL rewriting appends the session ID to the URL for every page that’s requested. The only problem here is that you must rewrite every link in your HTML code as well as those from servlet to servlet, or servlet to JSP.

The procedure for URL rewriting is quite simple and requires only the use of two methods found in the HttpServletResponse interface. These two methods, encodeURL() and encodeRedirectURL(), are used to append the session ID to the URL. This allows the server to track users as they move through your Web pages, but it requires that every URL be rewritten. The string returned by the methods will have the session ID appended to it only if the server determines that it’s required. If the user’s browser supports cookies, the returned URL will not be altered. Also, the returned URL won’t be altered if the application server is configured to not use URL rewriting. The format of the altered URL will vary based on different application server implementations; however, the common format will be the addition of a parameter, such as “sessionID=uniqueIDnumber”. The parameter name (in this case “sessionID”) is usually controlled through a configuration setting on the server. The value of the parameter (“uniqueIDnumber” in this example) is the unique session ID assigned by the server’s session manager and is a long series of letters and numbers. The following line of HTML code from a JSP creates a link to another JSP:

```
<A HREF="/products/product.jsp">Product Listing</A>
```

Clicking on this link would send the user to the product.jsp page. Using URL rewriting, the same code would be written as follows:

```
<A HREF="<%= response.encodeURL("/products/product.jsp")
   %>">Product Listing</A>
```

The returned string from the encodeURL() method would contain the session ID. On a Tomcat 3.2 application server, the result of this line of code would be:

```
<A HREF="http://www.yourservername.com/products/
   product.jsp;$sessionid$xxxx">Product Listing</A>
```

The xxxx would actually be a unique session ID generated by the server. The other method you can use for rewriting

sitraka

www.sitraka.com

URLs is the encodeRedirectURL(). It's used only in a servlet or JSP that calls the sendRedirect() method of the HttpServlet-Response interface. The following code is a standard redirection statement:

```
response.sendRedirect("http://www.yourservername.com/
  products/sale.jsp");
```

Using URL rewriting, the code would be:

```
response.sendRedirect(response.encodeRedirectURL(
  "http://www.yourservername.com/products/sale.jsp"));
```

The application server handles the encodeRedirectURL() method a little differently than the encodeURL() method; however, each method produces the same result.

You should now have a good understanding of how the session ID is tracked and matched to a session object on the server. The first step in using the session object is creating it. The method getSession() is used to create a new session object and to retrieve an already existing one. The getSession() method is passed a Boolean flag of true or false. A false parameter indicates that you want to retrieve a session object that already exists. A true parameter lets the session manager know that a session object needs to be created if one does not already exist. The following line of code demonstrates the use of getSession():

```
HttpSession session = request.getSession(true);
```

The getSession() method will return the session object. A new session object is created if one does not already exist. The server uses the session ID to find the session object. If a session ID is not found in a cookie or the URL, a new session object is created. You would probably use only the getSession() method with a true parameter at one point in your Web application. This would be the starting point of your site, possibly after the visitor has successfully logged in. Other servlets in your application should use the getSession(false) method. This will return a current session object or null. It does not generate a new session if one doesn't already exist.

A number of methods are defined in the Java Servlet specification. (A complete API can be found on http://java.sun.com.) The methods you'll use most often and the ones we'll focus on are:

- ***setAttribute(String name, Object value):*** Binds an object to this session using the name specified. Returns nothing (void).
- ***getAttribute(String name):*** Returns the object bound with the specified name in this session, or null if no object is bound under this name.
- ***removeAttribute(String name):*** Removes the object bound with the specified name from this session. Returns nothing (void).
- ***invalidate():*** Invalidates this session and unbinds any objects bound to it. Returns nothing (void).
- ***isNew():*** Returns a Boolean with a value of true if the client does not yet know about the session or if the client chooses not to join the session.

For an example in using sessions, we'll look at session management code that could be used for an online banking application that will allow customers to view their account information. The design of the application will follow the Model-View-Controller (MVC) architecture. The model, or data and business logic, will be represented by JavaBeans; the view will be through JavaServer Pages; and the control of the application will be handled by servlets. The ideas in these examples can easily be implemented in other types of Web applications.

An online banking application should have an HTML login page where the customer can enter a login name and password in a form. The form will submit (or post) the name and password to a login servlet. The first thing the servlet needs to do is verify the username and password. To stick with the topic at hand (sessions), we'll look only at the code needed to handle the session. After the customer has been verified, a Customer JavaBean can be created. The Customer bean will contain the basic information about this visitor and will be stored in the session. We want to create a new session object, but we also want to invalidate a session that may already exist. To do this, we need to retrieve the existing object (or create a new one) and check if it's a new session using the isNew() method. If it's not a new session object, we need to invalidate it using the invalidate() method. In the servlet, we can accomplish this with the following code:

```
HttpSession session = request.getSession (true);
if (session.isNew() == false) {
   session.invalidate();
   session = request.getSession(true);
}
```

"The process of placing an object into the session object is known as *binding*"

The first line of code generates a new session object, or retrieves an existing one. The second line sees if the session is new by checking the value from isNew(). A true tells you the session was just created; a false means this user already had a session and you need to invalidate it. One possible reason the user would have an old session is that he or she has two accounts and logged in on one, then tried to log in on the other.

You can now add the Customer JavaBean to the session for future use. The process of placing an object into the session object is known as *binding*. The Customer object can be bound to the session using the setAttribute() method as follows:

```
session.setAttribute("CustomerBean", Customer);
```

Since we're working on a bank's Web site, security is a priority. To be secure, every JSP and servlet needs to verify that this user is an authorized customer before displaying any information. To accomplish this, each servlet should contain code that looks in the session for a Customer object and sends any customers who do not have this object to a login page. The following code handles this:

altoweb

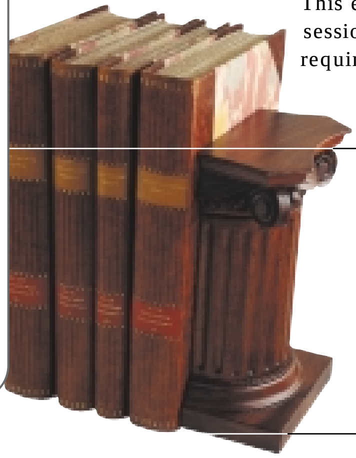
www.altoweb.com

```
Customer customerBean = (Customer)
  session.getAttribute('CustomerBean');
if (customerBean == null) {
  response.sendRedirect("https://www.yourservername.com
    /login.htm");
    return;
    }
```

The customer will have a valid Customer JavaBean in the session if he or she logged in properly. A getAttribute() on a name that does not exist in the session will always return null. Visitors with a null value need to log in, so we redirect them to a login page. This code should be placed at the top of the JSPs to prevent unauthorized use of the site. The code should also be placed in the servlets. Remember that the JSP creates the session variable for you, and in a servlet you must create it yourself. Keep in mind that this is just one way of handling security for a Web site. Some Web application servers will handle authentication and authorization for you.

This example is just a simple demonstration on session management. Your Web application may require more advanced security measures.

You may have noticed that the object returned by the getAttribute() method is cast into a Customer object. This is necessary for any object bound to the session. The object is stored in the session as an Object type. To use the object in your code, you must convert it (or cast it) back to the type of object it is.



As customers move through your site, they may wish to retrieve various pieces of information about their accounts. For example, they may be able to view their checking account balance by clicking on a menu option. Following the MVC architecture, the link would send them to a servlet that would verify who they are and then create a CheckingAccount JavaBean. This object would then be stored in the session using the setAttribute() method, then the servlet would send the customer to a JSP that uses the CheckingAccount JavaBean to display the information about the customer's account.

There may be times when you want to store something in the session other than a JavaBean, such as a text string or a number. You need to remember that you can only bind objects to the session. Text may be stored as a String object. You can put a number in the session as an Integer object. The following code demonstrates how to save a line of text and a number in the session:

```
session.setAttribute("text","A line of text.");
session.setAttribute("number", new Integer(10750));
```

Remember to cast the objects when they are retrieved:

```
String myText = (String) session.getAttribute("text");
   int myNumber = ((Integer) session.getAttribute
     ('number')).intValue();
```

Because the session is so easy to use, you may overuse it. Simple messages from a servlet to a JSP can be placed in the session as String objects, but this is not the most efficient way. Soon you'll find that your session is loaded with messages, and most of them are no longer needed. If you find that you're passing simple strings back and forth in the session, perhaps you should consider wrapping up those messages in a special JavaBean. This would keep the session more organized. The session objects for each user of a Web site are stored in memory on the server. Throwing unnecessary information into the session reduces the server's memory resources. Store only essential information in the session and use the removeAttribute() method to clean out objects after you're finished with them.

The use of the session will not only make it easier for you to program a site, but it should also help make the Web visit better for the user. The use of the beans and the session allows you to write JSPs that are customized for each user. For our bank scenario, we could use the information in the Customer bean to create personalized pages for each visitor. We can also use the bean to prepopulate forms for the customer. For example, you could use a JavaBean to store the results from a form that a visitor fills out to request information about services offered. If the user forgot to fill in a required field, you could use the JavaBean to store error messages from a servlet and then display them in the JSP. You could also populate the fields that the visitor just filled in instead of making the visitor fill out the form again.

"As Java developers, we have access to a powerful and robust session manager through the use of the HttpSession API"

The session is not intended to be used as a persistent place to store information about a visitor. Each visitor will be assigned a new session every time he or she logs in. A back-end database will be needed if you want to store information about individual users. The session should just be used to track the user during one visit to your site. In cases where you have a large Web site and are running multiple, redundant application servers, you'll need an application server that can handle sessions across servers. This is usually handled by placing session information into a database instead of local memory so each application server can access the information. And many of the commercial application servers will be able to do this for you.

Conclusion

The use of session tracking is an important design issue because of the complexity of today's Web sites. As Java developers, we have access to a powerful and robust session manager through the use of the HttpSession API. The session examples that we went over in this article cover the main capabilities of the session API. Learning all of Java's session management features will make your job as a Web developer easier and help you create a better experience for your Web site visitors.

AUTHOR BIO
Brian A. Russell is a software engineer for Priority Technologies, Inc., in Omaha, Nebraska.

brian.russell@prioritytech.com

borland

www.borland.com

Delivering a J2EE Application Suite

A case study

WRITTEN BY
KUNAL SHAH &
AJIT SAGAR

With the rapid adoption of J2EE has come the realization that more than just J2EE expertise is needed to successfully develop enterprise applications.

The following are critical to the success of J2EE projects:

- A well-defined component integration strategy
- The ability to select the right J2EE platform vendor
- Good understanding of internal and external expectations
- Evaluation of technical and nontechnical challenges and risks
- Availability of resources and tools
- Capabilities of the team
- Realizing that the best J2EE design practices have to be tailored to meet the needs of the specific project

The ultimate goal of any development project is to produce quality software that meets the requirements of the customer, and is on time and under budget. J2EE adds another dimension to this challenge. Distributed Java projects are highly componetized. Componentization has many advantages as well as challenges, the biggest of which is bringing it all together when different components have been built in parallel.

J2EE projects typically combine a variety of components built on EJBs, servlets, JavaBeans, JSPs, HTML pages, and the database. Each tier of an application is developed separately and follows a different "life cycle" in the development process. For example, a minor change in the underlying database schema or an EJB can cause serious ripple effects in the application, whereas changing a servlet or a JSP does not constitute a risk of the same degree.

This article captures our recent experience in successfully managing a six-month, 22,500 man-hour J2EE project in a startup organization. It focuses on engineering processes that had to be set up to support the development of enterprise software. While we borrowed from different development and design processes, we also had to evolve our own to custom-fit into our environment, as is normally the case with real-world projects.

The primary motivation for writing this article was to pen down the processes, pitfalls, and challenges of successfully completing a fairly complex J2EE project in a bootstrap environment. We describe a practical approach to release and change management for a successful J2EE project including setting up the environment; defining project teams, the tools and processes used; and J2EE-based design, development, and deployment.

The Environment

It goes without saying that the right level of J2EE expertise is a must to make a J2EE project successful. Equally important is having the correct environment for the team to be the most productive. The development environment typically includes:

- An IDE that supports Java development
- Configuration management tools and processes that support various distributed components
- A version-control system that supports Java applications
- A bug-tracking system
- Build processes and an overall release methodology
  - –The release methodology has to incorporate the fact that business components are built and released in a different manner than Web components built using servlets and JSPs.
  - –The dependencies of the business layer and the presentation layer have to be taken into consideration in the release process.

The release methodology in our project was customized to support the multitiered nature of J2EE architecture. The development and release of each tier was done separately so that a defect in the components produced by one team would have minimal impact on the rest of the team.

The development teams for this project had a couple of seasoned Java architects and developers as well as junior engineers. The end game was to develop a software product that was highly customizable and would adapt to different industry verticals. BEA WebLogic Server was used as the application server. Specific J2EE technologies used in this project are discussed later in this article.

The project consisted of five major independent applications that reflected real-world functionality. An important objective of the application suite was to ensure that these components could be run as independent products as well as

spirit soft

www.spirirsoft.com

an integrated suite. They were developed as different applications within the same application server.

The development team was split into five subteams; each team had between two and four developers and was responsible for one application. This turned out to be the optimum team organization for a company of less than 30 employees.

In all, three different types of environments were set up to support the release management process (see Figure 1).

Development Environment

The boxes in the left section of Figure 1 depict independent Windows 2000 desktops and laptops. Each developer's personal desktop was used as the development environment. It had access to a database schema that could be used for development purposes and was focused on a particular tier of the application – the JSP and HTML tier, the servlet tier, the EJB and business component layer, and the database layer. Each developer worked on a specific layer at a given time.

Integration Environment

The middle section of Figure 1 shows the integration environment. A single application server machine was set up to run all five applications within BEA's application server. The application server pointed to a dedicated database schema reserved for integration testing. Components developed separately in the development environment were brought together in the integration environment.

Staging Environment

The staging environment consisted of two application servers and a database server. This closely resembled the production. Once the application was integrated and packaged into an installable application, it was promoted to the staging environment.

Project Life Cycle

The project was broadly divided into a development phase, an integration phase, and a test phase, as is the norm. However, since we were in startup mode, it was crucial to expect and accept changes in direction in order to market and sell the product suite. While testing was started in parallel with the development phase, it was not until the integration phase that the entire application was available for testing.

Being a startup organization, an interesting distraction was to support ongoing sales activities during the entire development phase. Demos had to be built in short periods of time for the sales team and required the entire product suite to function in an integrated manner. Such unscheduled activities can potentially derail a well-planned development project. Instead of treating the demos as distractions, we used them as opportunities to develop prototypes for the yet undeveloped features. This enabled the sales team to present a more complete solution to the prospective client. Even as the demos proved to be a great advantage, there were many challenges. The top challenges were:

- To ensure that the development effort continued while part of the team was diverted to build the demo. Existing activities were reprioritized so that they would have minimal impact on the schedule.
    - To evaluate new requirements (generated from the demos) and their impact on the existing design, functionality, and schedule. If major changes had be made to the underlying data model or in the EJB layer, they were less likely to make it into the release. On the other hand, if new requirements could be implemented by changing servlets or JSPs, there was a greater chance that they would make it into the release.
- To roll back ad hoc enhancements into the development cycle.

Due to this continuous feedback, the team was able to redesign certain components to better reflect the requirements from the field. At least one engineer from each team was assigned to work on the demo. Putting together an entire demo was another interesting activity, since it was always a hybrid of well-designed features as well as those features that were just prototyped for the demo. J2EE is an ideal platform for managing a changing codebase because of its inherent support for decoupled components.

J2EE Technology Components

The project conformed to the standard three-tier J2EE architecture. The application suite was built using the following:

- JSPs, servlets, and JavaBeans formed the presentation layer.
- EJBs encapsulated the business logic.
- JMS was the messaging layer for the asynchronous exchange of business events.
- BEA WebLogic Server was used as the application server.
- BEA WebLogic Integration was used as the J2EE platform for business process management.
- Oracle 9.1 database housed the business data.
- Several third-party libraries were deployed to support charts, graphs, etc.

Change Management

We used Ant-based compile scripts to set up the integration environment. The staging environment was set up so that the application could be installed via the installer. Microsoft's Visual Source Safe (VSS) was used as the version control system.

During the development phase, the integration environment was built on a regular schedule. Earlier in the project, the frequency of a new build was once a week. The build frequency

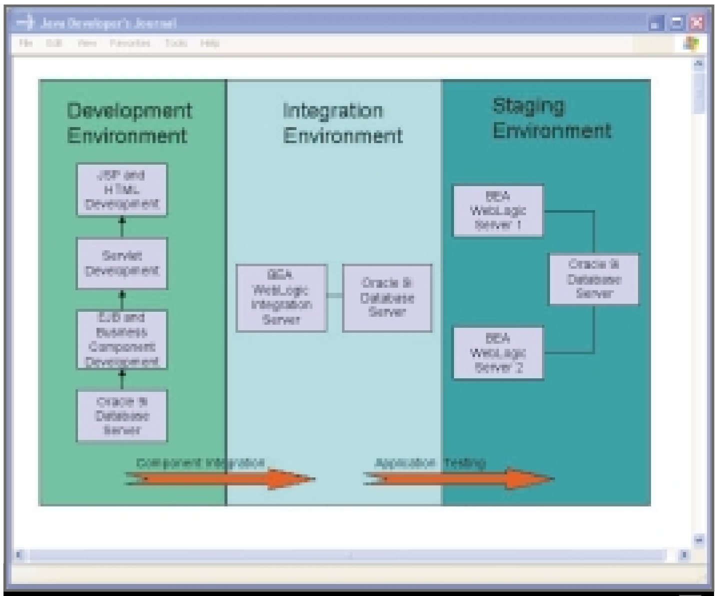


FIGURE 1 Hardware and software environment for the project

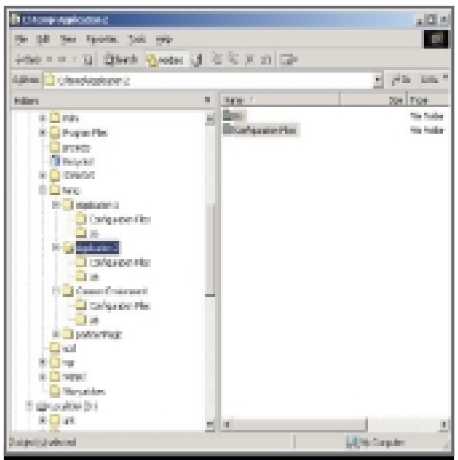


FIGURE 2 Directory structure

precise

www.precise.com

ibm

www.imb.com

increased to 3–4 builds a day toward the end of the release. The steps involved in rebuilding the integration environment were:

- Check out the latest code base from VSS for the entire product suite along with all necessary configuration files and incremental schema changes if applicable.
- Build the entire product suite using Ant scripts.
- Configure all the necessary properties files.
- Apply incremental schema changes to the database.

Since each application could be run as a standalone system, the following changes had to be made via configuration files to execute them within the same environment:

- By default, each application was set up with its own set of libraries, JAR files, classpaths, property files, etc. The applications had to be configured so that they shared a common environment. Thus a common configuration file had to be created that would be used by all the applications, and a common "lib" directory had to be created that contained the shared libraries and JARs.
- Application-specific EJBs and JAR files were local to each application.
- By default, each application pointed to a different schema. The properties files had to be updated so applications shared the same database.
- Some of the HTML files and JSPs had to be modified to provide an integrated user interface.

Figure 2 shows how the directory structure was set up to ensure that all common components were shared. In Figure 2, application 1 has its own set of classes, JARs, config files, JSPs, etc. It also shares the common classes, JSPs, JARs, etc., from the Common Environment folder.

Each component was developed and tested in its own environment. Once the key features were developed, they were integrated in a separate "integration environment." Once the applications were integrated, they were promoted to the staging environment. The defects found in this environment were reproduced and analyzed in the integration environment.

Figures 3 and 4 show the change management steps across various environments.

Change Management of Shared Components

Shared components implement functionality such as user management, common logger, and audit trail. Common functionality is shared across many applications. All EJBs, property files, JSPs, or database entities that support such common functionality have to be carefully managed since it impacts more than one application. Additional coordination is necessary to ensure that other teams can continue to be productive. The changes were completely tested in the integration environment before checking them into VSS. In other words, the next steps were followed to ensure the quality of the shared components:

- Obtain the most current code base from VSS.
    - Obtain the modified shared components from the developer's environment and overwrite the code base from VSS.
    - Compile the application using the integrated build script.
    - If the functionality works as expected, immediately check the modified components into VSS.

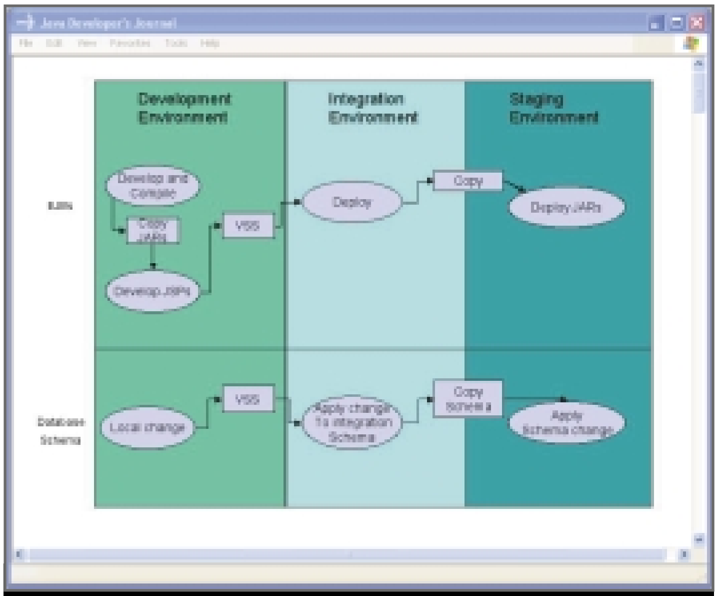


FIGURE 3 Change management across environments (business components)

Change Management of the Business Logic Layer

Sometimes changes in the EJB layer meant changing the presentation layer. JSPs and servlets may have to be fixed to reflect new changes in the EJB layer. Such changes were coordinated by the development team and implemented in the development environment. Developers working on the presentation layer used the updated EJB JARs and ensured that the presentation layer worked. All the changes were then checked in at the same time. Figure 3 shows the steps involved.

Change Management of the Database Schema

Typically, schema changes made to support new features are most likely to break the system. Schema changes made to fix defects must ensure that existing features are not impacted. However, they were discouraged in this project, and were made only if they impacted critical functionality. Change management of schema changes is shown in Figure 3.

Change Management of the Presentation Layer

Figure 4 shows the change management steps in the presentation layer. It can be seen that some changes are rolled forward from the development to the staging environment and some changes have to be rolled back from the staging to the development environment. Since the presentation layer has the least number of dependencies, it can be changed easily in any environment. During testing, it was sometimes easier to tweak a JSP in the test environment itself. These changes were then applied to the source in VSS. The challenge here was to be disciplined and roll the changes back into the development environment.

Tools and Processes

Various tools and processes have to be in place to support software development. Some of the key management aspects that must be addressed are time, process, version control, configuration, and defect. Being a startup organization, the team had to set up each and every process and tool from the ground up. The following are some of the criteria used to select these tools and processes:

- Should adequately support the requirements
- Be easy to use, configure, manage, and measure
- Be cost effective
- Be scalable to support a growing team for at least two years
- Have a relatively low learning curve for the team
- Be practical to use and not create additional work

Configuration Management

Configuration management controls and monitors change throughout the software development life cycle. Since

altova

www.altova.com

we had to branch out often to support ongoing demos, it was extremely important for the team to follow a consistent configuration management process. Visual Source Safe (VSS) was used for configuration management and version control. Every element of the project was under configuration management right from the get-go, including:

- Requirements documents
- Configuration files
- External libraries
- Source code

Defect Tracking

There are several defect-tracking tools available under public domain that work well in small organizations. A tool called codeCharge code generator was used to build the bug-tracking tool. CodeCharge generator allows a developer to quickly create a Web-based application in a variety of languages including ASP, ASP.NET, JSP, and ColdFusion. It supports all standard relational databases including MS Access. This tool enabled us to incorporate new features without much down time.

Build Process

Each developer set up his or her own build environment. The solutions ranged from using the built-in functionality of an IDE to using Ant and the basic batch script. Developers were not allowed to check in functional enhancements without ensuring that the entire application compiled and the new functionality worked in the local environment. A common build script using Ant was developed for each application. Thus, getting the most current version was as simple as obtaining the current code base and building it with the Ant scripts. The build scripts failed very rarely since the developers validated each and every code change.

Test and QA Process

You rarely have the luxury of a dedicated QA team in a startup environment. While the development team was busy building the new product, the domain experts were busy creating a functional test plan. A "war room" environment was set up for the entire development team for two weeks following the integration phase. With help from subject matter experts, two weeks were dedicated to testing the new product. Defects were documented and classified. Defects that prevented the execution of key functionality were classified as "show stoppers" and fixed immediately. The remaining defects were prioritized and either fixed following the test cycle or documented and scheduled for the next release.

Challenges and Compromises

Embarking on a new J2EE project is always challenging, especially in a small growing environment where the success or failure of the project depends on each decision – from selecting the right tools and setting up processes to making hard decisions and managing expectations. This project was no different. Some of the project management challenges were:

- ***Requirements Management:*** Since the sales team was able to provide feedback well into the development cycle, requirements were never really frozen. Each new requirement was carefully evaluated and prioritized. If a new requirement was deemed to be critical for the release, it would be further analyzed and the impact on the schedule and design evaluated.
- ***Testing and QA:*** A unique environment involving the entire team had to be created to test the product. The war room environment helped the team focus on the task at hand and iron out the rough edges.
- ***Expectation Management:*** Since this was a new team and each person brought in unique experiences from prior jobs, managing the team's expectations was a high priority. The team had to be reminded frequently that the integration environment was for integration testing, not development, and that changing requirements have a huge impact on the quality of the product, etc.
- ***Time to Delivery:*** As with most development projects today, time-to-market was the most critical metric for this project. Even though it was challenging, the project was completed on schedule by making certain compromises.
- ***Mentoring:*** The project plan had to include mentoring activities, given the varied levels of J2EE expertise in the development team. Since there was a learning curve, productivity metrics had to adjusted for the entire team.

To meet the goal of delivering a quality product on time, several compromises had to be made. Several performance-related issues were identified, but not resolved as this would have pushed the project beyond the release date. Portability was postponed to a future release. Compromises also had to be made when selecting tools and processes, since the bootstrap environment did not justifying buying commercial tools such as Rational Rose, ClearCase, WinRunner, and TestDirector. In some cases, functionality that should ideally be developed in the EJB layer was developed in the servlet layer to meet the time-to-market pressures.

Conclusion

J2EE projects can be completed on time by following the basic principles of project management. They can prove to be more challenging, as they target enterprise applications that require large integration efforts. Identifying potential risks and then taking quick actions to mitigate them keeps the project under control. This prevents small "fires" from destroying the entire project. Technical issues are relatively easy to resolve as compared to people and process issues. It's the responsibility of project management to get a buy-in from the stakeholders on the processes and tools to be used.

It's human nature to resist change, especially when you don't see value in the process. Using best practices as a guideline and making pragmatic decisions go a long way in making any project successful. Fortunately, J2EE is mature enough, so you can learn from other people's experiences.

kshah@controllingfactor.com

ajit@sys-con.com

AUTHOR BIOS

Kunal Shah is the director of product management at Controlling Factor. He is well versed in Java-based product management, ranging from small to large enterprise projects

Ajit Sagar is the J2EE editor of JDJ and the founding editor of XML-Journal. He is the director of engineering at Controlling Factor, a software firm in Dallas. He is well versed in Java, Web services, and enterprise technologies.

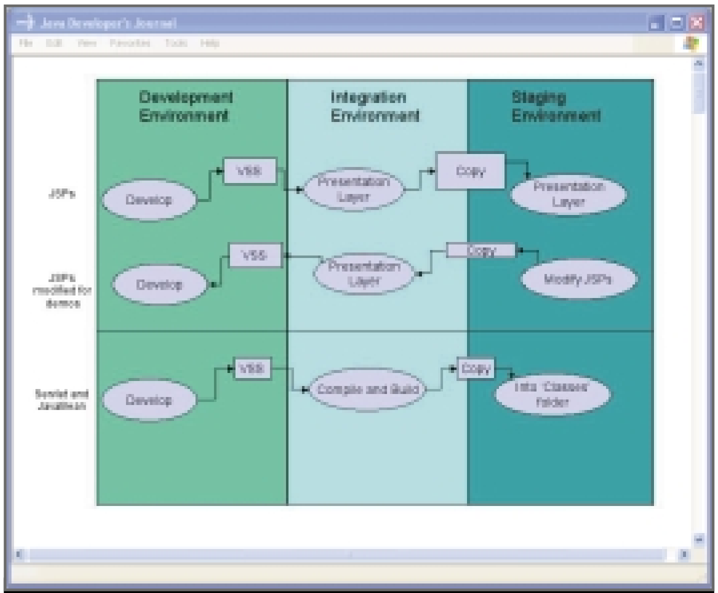


FIGURE 4 Change management across environments (presentation layer)

acceltree

www.acceltree.com

JASON BELL J2SE EDITOR

The 84% Rule

According the Standish Group, 84% of all IT-related projects are not delivered on time or within budget. Now when the world reads "IT-related projects," the automatic assumption is that the IT department is to blame.

Further investigation reveals the main reasons for failure: inadequate requirements and lack of client/user input. I've worked both extremes: I've written functional and technical specifications and the client has responded (via the project manager) that the specs were like "a sledgehammer to crack a peanut." If this is the client's mentality, you'll encounter trouble.

On the opposite side of the coin, I've received specs from on high that were completely unreadable or made no sense at all. When it came time to deliver, the client said, "Well, I actually meant this." My favorite was the client who said, "I don't understand what I've written."

Are we programmers that unapproachable? Is it our armpits? No, it's a lack of basic communication. Managers feel they want to try and talk "techie" to us so we might understand what they're saying. What we really want is an English (or insert your native tongue here) description of their aim and vision and some basic requirements. Then we can come to some agreement on functional specifications.

What worries me is that when managers of different companies talk together, a common question will crop up: "Does your IT department deliver on time?" The response will usually be, "No, we leave them to it; we dare not approach them!" Have I hurt your feelings yet? No? Good.

Perhaps it's time to teach. The word *teach* is an interesting word. The usual dictionary references are fine, but if you look at the Hebrew definition it means, among other things, to "shoot arrows" (Strong's number 03384). You need to be pretty direct when trying to get your point across.

Not everyone is a teacher so it's always handy to have someone who can present the idea quickly and concisely – your managers will have more faith in you and your team. Get to know your team, know each member's strengths and weaknesses. Yes, they're all great programmers, but who communicates the ideas the best? Who is the visionary? Who keeps everyone focused on the light at the end of the tunnel?

If you don't use a software development methodology, you should start; it doesn't matter if it's UML, Extreme Programming, etc. You don't have to take the whole method as gospel, but the more you can plan out and see the risks and communicate those risks to management, the better. Highlight the major problems and sort them out straightaway.

Another thing I've found handy is to deliver the project to the client frequently throughout the project's timeline. I started doing this after talking to other developers. They would create the application with limited or no functionality and keep delivering it on a regular basis. This creates a sense of trust with the client that something is actually happening. The client doesn't have to wait until the last quarter of the project to actually see something half working.

I've had to look at my track record and say, "Hey, I could have done that better." It's all about personal as well as team development. Over the years I've made a mess of estimating projects – sometimes it was due to bad requirement specs and sometimes it was me. Our office now has a motto – "Woolly specs = woolly deadlines."

Feel free to let me know how you get on. Until next month.

References

- *The Standish Group:* www.standishgroup.com
- *Strong's Lexicon:* http://bible.crosswalk.com/Lexicons/Hebrew/

jasonbell@sys-con.com

AUTHOR BIO

Jason Bell is a programmer and senior IT manager for a B2B Web portal in York, England. He has been involved in numerous Web projects over the past five years, the last two of which have been servlet-based.

J2SE INDEX

esri

www.esri.com

Combating the 'Object Crisis'

The foundation for Java proficiency

WRITTEN BY
JACQUIE BARKER

Java's phenomenal success as an enabler of enterprise-wide, Web-deployed applications has compelled countless organizations and individuals to seek Java proficiency. Many are drawn like moths to a flame, and in fact go "down in flames." They're ill-prepared to harness Java's power as an object-oriented programming language due to a basic lack of understanding of object concepts.

Many professional software developers schooled only in procedural programming languages are now looking to make the transition from procedural to OO programming with Java. Motivated by career survival, they seek a "quick fix" – a single book or course that can transform them into Java-proficient software engineers.

In addition, the relative shortage of Java-trained professionals has led organizations to attempt to mass-produce Java talent by retooling their in-house programming staff in Java. Eager to do so in the quickest way possible, they fall prey to the notion that sending competent programmers to Java training – especially C++ programmers, who are presumably already proficient with objects – will produce instant "Java savvy."

What these individuals and organizations often don't realize is that:

• ***Most Java training does only a cursory job of explaining object concepts.*** After a perfunctory lesson on what objects and classes are, Java instructors usually dive into the details of Java syntax without giving participants a big-picture appreciation for the nature of objects. Topics key to harnessing the full power of an OO language – e.g., encapsulation, information hiding, object collaboration, overriding, polymorphism – are merely touched upon in passing, if at all. Students go through the motions of writing Java code "snippets" but often come away without any knowledge of how to structure a software application from the ground up to make the most of Java's OO nature.

• ***Not all C++ programmers are object savvy.*** Many software engineers who adopted C++ years ago were "born-again" C programmers who saw C++ as a "better C." They weren't necessarily compelled to learn the object paradigm beyond a superficial level because they were able to successfully write procedural C++ code upon merely learning C++ language syntax. (Despite many syntactic similarities between the two languages, Java is arguably more true to the object paradigm – it provides fewer "back doors" with which a programmer can escape the rigor of objects.) Those C++ programmers who never learned objects properly don't have a leg up learning Java, but organizations unfortunately assume they do.

• ***The prevalence of drag-and-drop Java IDEs exacerbates the situation.*** While such tools allow those without object know-how to craft OO user interfaces, these UIs sit on top of a non-OO infrastructure. Like the Hollywood facade of a town in the Old West, an application built with such an IDE may appear to be object-oriented. Scratch the surface, however, and you'll see the reality – there is a noticeable discontinuity between the OO front end and a decidedly non–OO back end. This leads to brittle applications that are difficult to extend and maintain.

The "object crisis" is by no means insurmountable. By observing a few basic guidelines for how best to retool with objects in general and Java in particular, you can quickly be off to the right start:

- Invest in object training *before* Java training: it's like learning how to hold a golf club properly before strategizing how to play a particular golf course. But choose wisely. Make sure that the object training you select doesn't teach objects in isolation – it should also illustrate how to bridge the gap between object models and Java code.
- Craft your own Java code using a bare-bones IDE or simple text editor to master the OO aspects of Java before relying on a drag-and-drop GUI builder to churn out code automatically.
- Engage a Java-proficient object mentor to work with a fledgling Java team throughout the project life cycle.
- Tackle a reasonably small project first – don't attempt to conquer a major enterprise-level application. Ideally, cut your teeth on an in-house project versus a project-for-hire for a key client.

The object paradigm is intuitive and powerful...nonetheless, mastery of objects doesn't happen automatically by virtue of learning Java syntax. An up-front investment in learning objects properly will pay for itself numerous times over in terms of the quality, maintainability, and robustness of the resulting Java applications.

AUTHOR BIO
Jacquie Barker is a professional software engineer, author, and adjunct faculty member at The George Washington University. Her book, Beginning Java Objects, *is focused on conquering the object crisis by teaching fundamental object concepts side by side with beginning Java syntax (www.objectstart.com).*

jjbarker@objectstart.com

macromedia

macromedia.com

Java Design

Using interfaces and abstract classes to create flexible code

Part 2 of 2

WRITTEN BY
MICHAEL BARLOTTA

In Part 1 (*JDJ*, Vol. 7, issue 6) we looked at the Java class as a type. Although it's easy to think of the class name of our Java class as its type, the interfaces it implements and the superclasses it extends can also be viewed as its types.

In Part 2 I'll explore using interfaces and abstract classes to achieve flexibility with a real-world example that implements the Data Access Object (DAO) pattern. I'll also quickly look at the abstract classes and interfaces in the Java arena.

Putting It to Real Use – the DAO Pattern

Data can be stored in different persistent data sources. These include relational databases as well as flat files, XML documents, LDAP, and legacy systems. Each data source requires a different way of getting a connection to it as well as various ways of retrieving, adding, updating, and removing data. With the DAO pattern you can separate the data source access and encapsulate interaction with the data from objects that use the data.

In any given application we may have a defined business entity called Customer. Rather than code all the data access logic in the Customer object, we'd want to separate this because the business that needs the application may store data for its customers in different databases. Some of the data may even be in legacy systems that require lots of code written with a proprietary API to get the data out. Keeping the data access logic out of the Customer business object makes the code cleaner and allows the developer to focus on the business needs instead of how to get the data.

Another practical use for the DAO pattern would be a product company that wants their application to work with multiple relational databases, allowing customers to use the DBMS of their choice. In either case we would want to use the DAO pattern to encapsulate data access from the rest of our application.

We can create a flexible implementation of this pattern using an abstract class and an interface. For example, a company has an application that needs to get customer data from either an Oracle or a Microsoft SQL Server DBMS. Because the SQL syntax for the Oracle and the Microsoft DBMS are not compatible, we'll need to write a separate class that can access the customer data from either database. These classes are represented by the OracleCustomerDAO and the MSSQLCustomerDAO. Because the set of methods for each of these objects is the same (get, add, update, delete) but they don't share any common implementation, we'll define an interface CustomerDAO as follows:

```
 public interface CustomerDAO {
  public CustomerDatagetCustomer
   (String id);
  public StringaddCustomer
   (CustomerData cd);
  public booleanupdateCustomer
   (CustomerData cd);
  public boolean deleteCustomer(String
   id);
}
```

The OracleCustomerDAO and the MSSQLCustomerDAO would implement each method using the specific SQL syntax of the DBMS to perform the operations.

The CustomerData object encapsulates all the data about our Customer. This class typically uses Java fields to hold the data and has no methods (other than accessor get/set methods). This is an example of the Value Object pattern.

Next we need to consider how to allow the application to get the proper CustomerDAO implementation without each class that needs customer data trying to determine which database is in use. Since this is a common operation, we can place that logic in a separate class that will then create the proper CustomerDAO object. This is called a *factory*.

Since the Oracle and Microsoft databases require different classes for each business entity we want to access (e.g., Customer, Order, etc.), we'll create a separate factory for each. The OracleDAOFactory class will have a getCustomerDAO method on it that will return a copy of the OracleCustomerDAO object.

```
public class OracleDAOFactory extends
DAOFactory {
public CustomerDAO getCustomerDAO() {
  return new OracleCustomerDAO();
}
…
}
```

The MSSQLDAOFactory has a similar method. Both implementations of the method return an object reference of the type CustomerDAO. This allows the business object, e.g., CustomerBean, to use the object to get customer data through the interface without caring which database the data is coming from.

The last thing we need to do is hide the Factory implementation from the business object. We can do this using a generic abstract class, DAOFactory. This class can implement the code to determine which data source to use and be the superclass for the OracleDAOFactory and the MSSQLDAOFactory. The code to determine which data source is in use is not shown but we could find out by reading in data from a Java properties file or an XML file, or looking it up in a JNDI Naming Service.

The abstract class, DAOFactory, can also define an abstract method, getDAOFactory, that's responsible for returning the correct factory object. It's the job of the subclass factory object to create the correct CustomerDAO for the given data source, as we saw earlier.

```
 public abstract class DAOFactory {
// abstract getXXXDAO methods
public abstract CustomerDAO
getCustomerDAO();

// getDAOFactory
public static DAOFactory
```

capella

www.capella.com

```
getDAOFactory() {

String factoryClass = "":

// determine which factory class
to use
…

Class _class = Class. forName
(factoryClass);

Object _object = _class.
newInstance();
 if (_object instanceof
DAOFactory) {
   factoryInstance = (DAOFactory)
_object;
 }
 else {
   // throw Exception
 }

 // Need to handle the
 // ClassNotFoundException
 // InstantiationException
 // IllegalAccessException

return factoryInstance;
}
 }
```

Our business object CustomerBean can now use either an Oracle or a Microsoft database to get, add, update, or delete a customer. The following code shows how a business object might use these objects to delete a customer:

```
String customer_id = "100";
DAOFactory df =
   DAOFactory.getDAOFactory();
CustomerDAO cust =
   df.getCustomerDAO();
cust.deleteCustomer(customer_id);
```

In this example we're writing to a type instead of a specific implementation. The code doesn't depend on the database that's in use and the application can quickly be changed to use new databases, such as Sybase. A new CustomerDAO class will need to be implemented using the Sybase SQL syntax to access customer data, and a Sybase version of the DAOFactory will need to be written; however, all the business objects that use the customer data won't require any changes.

Wrapping It Up

We've looked at how to write code to a type rather than to an implementation and saw how this can create a tremendous amount of flexibility in our applications. This is because the type determines what the object can do, and the implementation determines how the object does it.

The last thing to consider is how to determine when to use an abstract class and when to use an interface. Interfaces allow classes that don't share any implementation hierarchy (inheritance) to be grouped together and still share a type. However, when we use an interface, we don't get any implementation reuse as we do in the CustomerDAO interface.

When we use the abstract class as a supertype we get implementation reuse that doesn't need to be duplicated across multiple classes. For example, the DAOFactory can share the implementation code that determines which database to use. Using an abstract class, however, locks our class into an inheritance hierarchy and prevents other classes that already extend a particular class from sharing the type. It also prevents classes that extend the abstract class from being able to extend other classes.

This was not a major factor in the DAO pattern implementation but is usually a concern in other designs. A final consideration is that adding additional methods to the interface breaks all the classes that implement the interface, because each class is required to implement the new method, while adding methods to an abstract class can be done without affecting any subclasses.

References

- Gosling, J., Joy, B., and Steele, G. (1996). *The Java Language Specification*. Addison-Wesley.
- Flanagan, D. (2002). *Java in a Nutshell*. O'Reilly.
- Alur, D., Crupi, J., and Malks, D. (2001). *Core J2EE Patterns: Best Practices and Design Strategies*. Prentice Hall PTR.
- *J2EE BluePrints:* http://java.sun.com/blueprints/
- (Image and code) "Reveal the magic behind subtype polymorphism": www.javaworld.com/javaworld/jw-04-2001/jw-0413-polymorph.html
- "A primordial interface?": www.javaworld.com/javaworld/jw-03-2001/jw-0309-primordial.html
- "Thanks type and gentle class": www.javaworld.com/javaworld/jw-01-2001/jw-0119-type.html

mike.barlotta@aegis.net

Author Bio

Michael Barlotta is the director of technology at AEGIS Inc. (www.AEGIS.net). He's a Sun Certified Java programmer and is a recognized as an expert on Jaguar CTS (EAServer), mentoring clients and speaking at conferences on the topic. Mike is the author of several books including Taming Jaguar and Jaguar Development with PowerBuilder 7 (both by Manning Publications).

SYS-CON MEDIA

PUBLISHER, PRESIDENT, AND CEO
FUAT A. KIRCAALI fuat@sys-con.com
COO/CFO
MARK HARABEDIAN mark@sys-con.com
VICE PRESIDENT, BUSINESS DEVELOPMENT
GRISHA DAVIDA grisha@sys-con.com

ADVERTISING
SENIOR VICE PRESIDENT, SALES AND MARKETING
CARMEN GONZALEZ carmen@sys-con.com
VICE PRESIDENT, SALES AND MARKETING
MILES SILVERMAN miles@sys-con.com
ADVERTISING SALES DIRECTOR
ROBYN FORMA roybn@sys-con.com
ADVERTISING ACCOUNT MANAGER
MEGAN RING megan@sys-con.com
ASSOCIATE SALES MANAGERS
CARRIE GEBERT carrieg@sys-con.com
KRISTIN KUHNLE kristen@sys-con.com
ALISA CATALANO alisa@sys-con.com
LEAH HITTMAN leah@sys-con.com

EDITORIAL
EXECUTIVE EDITOR
NANCY VALENTINE nancy@sys-con.com
EDITOR
M'LOU PINKHAM mpinkham@sys-con.com
MANAGING EDITOR
CHERYL VAN SISE cheryl@sys-con.com
ASSOCIATE EDITORS
JAMIE MATUSOW jamie@sys-con.com
GAIL SCHULTZ gail@sys-con.com
JEAN CASSIDY jean@sys-con.com
ASSISTANT EDITOR
JENNIFER STILLEY jennifer@sys-con.com
ONLINE EDITOR
LIN GOETZ lin@sys-con.com

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

WEB SERVICES
WEBMASTER
ROBERT DIAMOND robert@sys-con.com
WEB DESIGNERS
STEPHEN KILMURRAY stephen@sys-con.com
CHRISTOPHER CROCE chris@sys-con.com
CATALIN STANCESCU catalin@sys-con.com

ACCOUNTING
ASSISTANT CONTROLLER
JUDITH CALNAN judith@sys-con.com
ACCOUNTS RECEIVABLE/COLLECTIONS SUPERVISOR
KERRI VON ACHEN kerri@sys-con.com
ACCOUNTS PAYABLE
JOAN LAROSE joan@sys-con.com
ACCOUNTING CLERK
BETTY WHITE betty@sys-con.com

SYS-CON EVENTS
VICE PRESIDENT, SYS-CON EVENTS
CATHY WALTERS cathyw@sys-con.com
CONFERENCE MANAGER
MICHAEL LYNCH mike@sys-con.com
SALES EXECUTIVES, EXHIBITS
MICHAEL PESICK michael@sys-con.com
RICHARD ANDERSON richard@sys-con.com

CUSTOMER RELATIONS
CUSTOMER SERVICE MANAGER
ANTHONY D. SPITZER tony@sys-con.com
CUSTOMER SERVICE REPRESENTATIVE
MARGIE DOWNS margie@sys-con.com
JDJ STORE MANAGER
RACHEL McGOURAN rachel@sys-con.com

ibm

www.ibm.com

Creating a Custom Launcher

written by john chamberlain

say goodbye to java.lang.NoClassDefFound

java.lang.NoClassDefFound

The most frustrating and error-prone aspect of Java for the average user is starting a Java program. The monumental confusion of batch files, scripts, and command-line cut-and-paste that's necessary to start a Java program using the default launcher is an ongoing problem area even for veteran developers.

This article shows how you can wipe away the whole mess and easily write custom launchers for your applications. A custom launcher makes startup as simple as point-and-click and can be the difference between a program appearing professional or appearing unusable. The specifics on making launchers for all the major platforms are also covered.

A long-time barrier to user acceptance of Java has been the confusing and unfriendly default launcher. It requires long strings of command-line arguments before it will start a Java program. Both users and software developers constantly receive the java.lang.NoClassDefFound exception and other errors. Sun's noble response to this problem is to provide a public API, the invocation interface, that can be used to start the JVM and accompanying Java program. Although a few commercial applications use this API, it's woefully underutilized overall. I'll show how easy it is to create a custom launcher and provide templates that you can use to automate start-up of your Java programs. There's even a generic configurable launcher that's ready to run, no compilation necessary.

I focus on the three major desktop platforms: Windows, Mac OS, and Unix. Each platform has its own quirks, but using a custom launcher brings benefits that are common to all three, such as:

- Program startup is easier and more reliable.
- Software identification and branding are better.
- Greater customization and VM control is possible.

Given that a commercial-quality launcher requires only about 200 lines of code and templates are provided, there's no reason you can't get started today.

How Java Programs Are Launched

The complexity of launching a Java program is largely due to Java being an interpreted language. This makes startup and shutdown a multistep procedure (see Figure 1).

As the figure suggests, any native executable can start a Java program. A Java launcher is a native executable dedicated solely to starting Java programs. The most commonly used launchers are the ones Sun supplies in the /bin directory of the Java runtime distribution. In the case of the Windows platform, these programs are "java.exe" and "javaw.exe". The former opens two windows: a console that receives System.out/err and output from the launcher and the Java window itself. The latter, a windowless launcher, opens only

parasoft

www.parasoft.com

"Once you understand the Java life cycle you're ready to code a launcher"

the Java window. On J2SE/EE platforms the virtual machine is implemented as a dynamic link library that's also in the /bin directory. On Windows it's called "java.dll", on Unix "java.so". Loading the VM equates to loading this DLL.

Users specify options to the VM in two ways. They can put the options on the command line to the launcher and/or define environment variables with the desired settings. One of the options, the startup class, can only be specified on the command line. This bifurcation of the execution configuration is a common source of confusion that can be eliminated by using a custom launcher.

When the virtual machine has finished running the main() method of the startup class, the launcher calls destroy() on the VM to free any detachable resources and then exits. Note that there's no way to unload a VM once it's been loaded. This makes no difference to a launcher since it will exit as soon as the Java program is done; however, for a native application that embeds a VM, such as a browser, it means there's a permanent commitment of memory that can't be reclaimed.

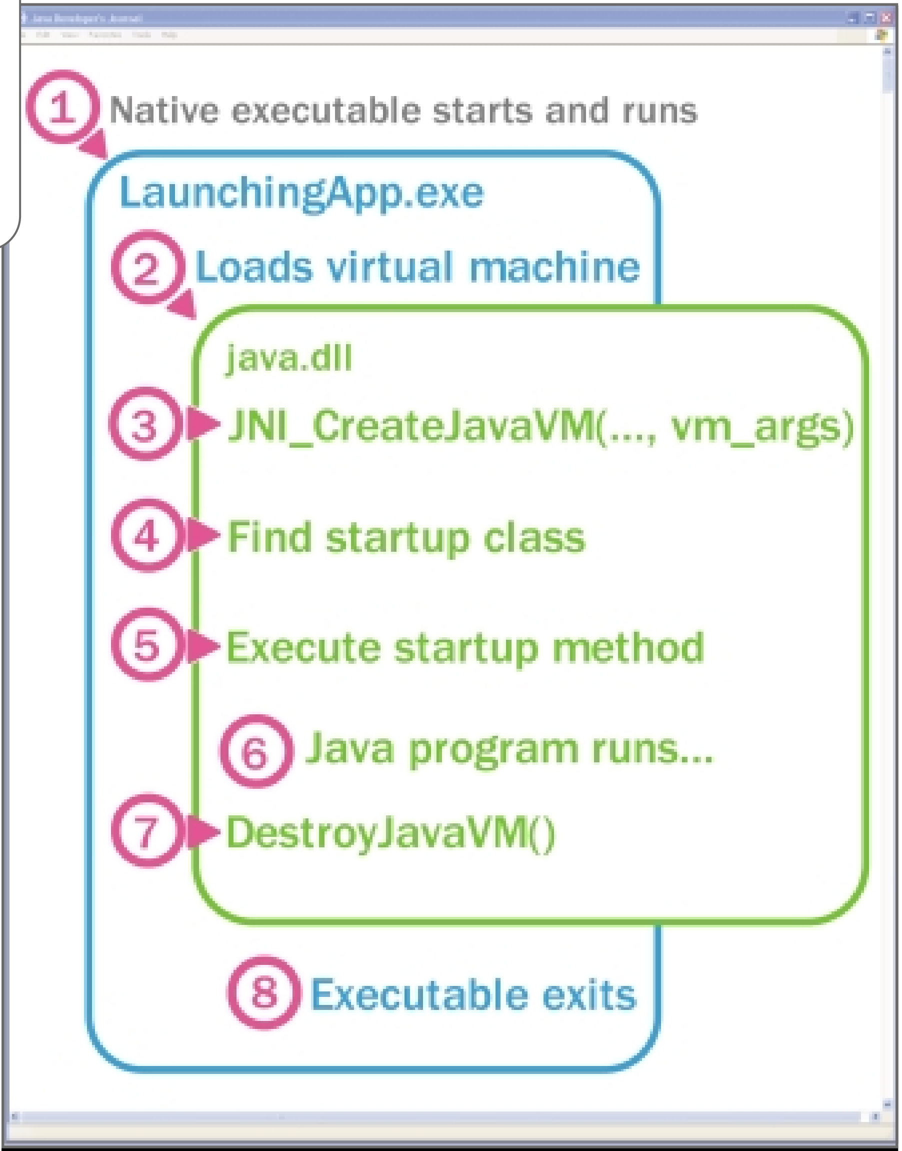


FIGURE 1 The Java life cycle

Nuances of Creating a Windows Launcher

Once you understand the Java life cycle you're ready to code a launcher. Be aware that some of the generic code examples floating around on the Web and in books, such as *The Java Native Interface* by Sheng Liang (see Resources), won't work on a platform such as Windows without changes. The working example in C++ for Windows illustrates some of the nuances (see Listing 1).

First, use a WinMain() entry point as you would for most Windows applications. Also, you need to prototype CreateJavaVM() to use the stdcall calling convention by type-def'ing it as a pointer to CALLBACK. These are Windows-specific requirements. Another platform-specific nuance is loading the VM DLL. The most reliable way to load the VM is by an explicit call to LoadLibrary:

```
HINSTANCE hJVM = LoadLibrary(sJVMpath.c_str());
```

First determine the path of the JVM's DLL and then explicitly load it. This differs from the example in *The Java Native Interface*, which uses implicit loading. The problem with implicit loading is that it makes assumptions about the location of the DLL that might not be true for all environments. By explicitly loading the JVM you can place it anywhere you like in your distribution and verify that it's really there before attempting to load it. Once you load the JVM, obtain a function pointer to CreateJavaVM() by using the kernel call GetProcAddress() and then calling that pointer to start the VM.

The next nuance in the listing is that the separators used in the startup class identifier are slashes, not dots. So in the listing the startup class is "javabunny/JavaBunny", not "javabunny.JavaBunny". This is because FindClass() is a virtual machine call and the virtual machine internally uses the slash as its package separator. By the way, the example hard codes the startup class (and other values). This may be appropriate for a shrink-wrapped product release, but in a development environment you'll probably want to pull this value from a configuration file. Later, I'll describe a more generic template that does this.

The example determines the startup method ID by using the JNI call GetStaticMethodID(). This call requires the method name ("main") and the type descriptor "([Ljava/lang/String;)V". This type descriptor means the method takes an array of strings as an argument and has a return type of void. For more information on type descriptors see *The Java Virtual Machine Specification* (see Resources). Notice that when you create a custom launcher you're not restricted to using a static void

engenuity

www.engenuity.com

A FEW WORDS ON JAVA WEB START

Java Web Start is an attempt by Sun to ease application deployment and execution. After installing Web Start on your system, you can click a link in a Web page to download and launch a Java application. Once it's downloaded, you can use Java Web Start's "Application Manager" at any time to execute the application.

One problem with Web Start is that you have no control over the Web Start interfaces, which have Sun all over them, so whatever product branding you might have gets stuck behind Sun's style and message. Along the same lines, your application icon is not native and is accessible only from inside the Application Manager, another Sun production. Also, your application does not have full functionality because it operates in a security sandbox similar to that of an applet. On top of this, JWS has had some stability problems.

Overall, Java Web Start has some interesting and attractive uses, but it's not a replacement for a custom launcher.

method called "main". You can start with any method at all, even an instance method or constructor.

The last tricky point of a launcher is hidden behind the following line at the end of the listing:

```
jvm->DestroyJavaVM();
```

This statement looks like optional cleanup added as an afterthought to program execution. Not true! If the Java program is multithreaded, it will still be executing during this call. For example, if a Swing program runs and its main method exits, this line will execute and block until all non-daemon threads have completed. This blocking behavior makes it critical that you include this line. If you omit it, the program will exit as soon as the main thread terminates, even if other threads (like the event loop of your GUI) are still running.

Launcher Configuration

In Listing 1 I hard code some of the key parameters such as the startup class. Notice, however, that none of the paths are hard coded. This is part of the beauty of a custom launcher – all the paths are relative, so you can drag the application folder to another drive (or computer) and it will run flawlessly. Try doing that with a batch file. Listing 1 always uses a JRE located in a subfolder of the application folder. By distributing a JRE with your application, like this, you guarantee runtime compatibility and make your application totally independent of the user's environment. The extra disk space used by adding yet another JRE to the user's disk drive is meaningless compared to the increased reliability. When writing your own launchers you may want to use different directory layouts than the one in Listing 1. As long as all the paths are relative to the native executable's location, you're fine.

"As long as all the paths are relative to the native executable's location, you're fine"

Resource paths can be made flexible enough that they don't need to be configured, but some values will need to be configurable outside of the launcher, especially in an oft-changing development environment. These include:

- The startup class
- The class path
- Special VM parameters such as "-verbose"

The best way to specify these parameters is to load them from a configuration file located in the same directory as the launcher executable. (The source code for this article can be downloaded from www.sys-con.com/java/sourcec.cfm and includes code for a launcher that configures itself this way.) By using a resource editor to replace the icons in this launcher's binary with your own, you can use it repeatedly for all your applications without ever needing to compile.

Mac Launchers

In the Macintosh universe, life is much easier. Java development on the fruit boxes is divided into two scenarios: OS X and pre-OS X ("Classic Mac OS"). OS X has strong Java support compared to Classic Mac. For example, Classic Mac supports only 1.1.8, so for many developers it will be irrelevant. Swing support on Classic Mac is available if the user downloads and installs MRJ 2.2.5, but for anything more recent like J2EE, forget about it.

If these restrictions don't faze you, create a native launcher on Classic Mac by using Apple's old native toolkit called JDirect (don't confuse this with the obsolete "J/Direct" that worked with Microsoft's J++). A much easier way to create a clickable icon, however, is to use a special Apple tool called "JBindery". This tool creates a distribution that so closely resembles a native application that writing a native launcher is unnecessary. You can completely configure your distribution package using JBindery, including defining security settings and the appearance of the Java window. When you're done, use ResEdit to add a custom icon to the package and it's ready to run. Apple considers Mac Classic, JBindery, and this whole methodology obsolete, but if you want to support the many users who are sticking with OS 9.1/2, it's your best option.

The new Mac world is all OS X. In OS X the application layer is called "Cocoa" and you access it with Objective-C. Is that retro or what? Despite how weird it sounds, Java support is excellent because an interface called the "Java Bridge" wraps the Java Native Interface (including the invocation interface) and makes a seamless connection between your native code and Java code.

As with the Classic OS, writing a native launcher is unnecessary, since Apple has provided a great bundling tool, MRJAppBuilder. If your Objective-C skills are a little rusty and you're working solely in Java, the best approach is to use MRJAppBuilder. Apple has designed this bundler especially for packaging Java applications. Note that the bundling framework the tool uses is the standard way to deploy all Cocoa applications, not just Java applications. This enlightened approach to application distribution means that on OS X, a bundled Java application is externally indistinguishable from an Objective-C application and behaves in all ways like a native executable.

The powerful capabilities of the bundlers (JBindery for Mac Classic and MRJAppBuilder for OS X) eliminate the need for a custom launcher on the Macintosh unless you're doing something offbeat such as starting from an instance method. If you really

jinfonet

www.jinfonet.com

need to go native on the Macintosh, the article's download package has some code examples that will get you started. Otherwise, stick with the bundlers and you can sit back and laugh at the PC programmers while they fiddle endlessly with batch files.

Unix Launchers

Unix (or Linux) is the inverse of OS X – it has no explicit support for Java or even for native application packaging. For example, on Unix desktops the icons live separately from their applications and the relationships between them are managed by configuration files or scripts. Issues like compiling icon resources into the launcher binary don't exist under Unix. This means an easy and reliable startup mechanism is more a function of your installation script than anything else.

Even so, a custom launcher still has many benefits under Unix. For example, in a process listing, the Java command line is usually so long it gets truncated, and on a server machine running multiple VMs it can be a pain to identify which process is which. You can create a custom launcher that simplifies and shortens these startup commands and thus make the process listing more meaningful.

One of the advantages of Unix's simplicity is that its launcher code is the easiest of all the platforms. The basic Unix launcher is the same as the Windows example shown in Listing 1 without the Windows-specific type conversions and Windows configuration issues (see the download package for an example). Another advantage is that a Unix launcher will generally work in any Unix environment as long as it's recompiled – once again, something that the installation script manages.

The disadvantage of this simplicity as compared to other OSs is that you are more or less obliged to use scripts of some sort even if you do implement a custom launcher. Good thing Unix has such great scripting capabilities.

Conclusion

By mastering the art of creating custom launchers for your Java applications, you can ramp up their convenience, professionalism, and reliability. The ease of creating launchers along with the use of configuration files makes them ideal for use in development environments as well as in release distributions. Do yourself a favor: learn to code a launcher and say goodbye to java.lang.NoClassDefFound.

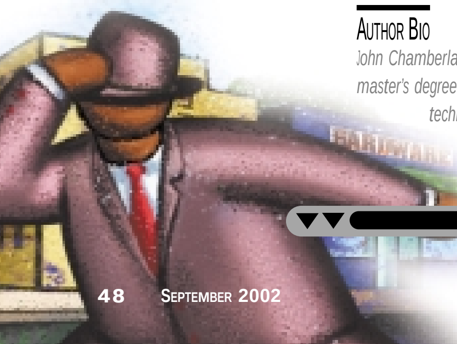
Resources

- Liang, S. (1999). *The Java Native Interface: Programmer's Guide and Specification*. Addison-Wesley.
- Lindholm, T., and Yellin, F. (1999). *The Java Virtual Machine Specification*. Addison-Wesley.

Author Bio

John Chamberlain is a consultant in the Boston area. He holds a master's degree in computer science, is a frequent contributor to technical journals, and has been a speaker at JavaOne. (http://johnchamberlain.com)

jcpublic@attbi.com



Listing 1: Typical Windows launcher for a 1.2 or later VM

```
#include <windows.h>
#include <jni.h>
#include <string>
using namespace std;

void vShowError(string sErrorMessage);
void vShowLastError(string sErrorMessage);
void vDestroyVM(JNIEnv *env, JavaVM *jvm);
void vAddOption(string& sName);

JavaVMOption* vm_options;
int mctOptions = 0;
int mctOptionCapacity = 0;

boolean GetApplicationHome(char *buf, jint sz);

typedef jint (CALLBACK *CreateJavaVM)(JavaVM
   **pvm, JNIEnv **penv, void *args);

int WINAPI WinMain(HINSTANCE hInstance,
   HINSTANCE hPrevInstance, PSTR szCmdLine,
   int iCmdShow) {

   JNIEnv *env;
   JavaVM *jvm;
   jint jintVMStartupReturnValue;
   jclass jclassStartup;
   jmethodID midStartup;

   // Path Determination

   // --- application home
   char home[2000];
   if (!GetApplicationHome(home, sizeof(home))) {
      vShowError("Unable to determine \
        application home.");
   return 0;
   }
   string sAppHome(home);
   string sOption_AppHome = "-Dapplication.home="
      + sAppHome;

   string sJREPath = sAppHome + "\\jre";

   // --- VM Path
   string sRuntimePath  = sJREPath +
      "\\bin\\classic\\"; // must contain jvm.dll
   string sJVMpath = sRuntimePath + "jvm.dll";

   // --- boot path
   string sBootPath = sJREPath + "\\lib";
   string sOption_BootPath =
      "-Dsun.boot.class.path=" + sBootPath;

   // --- class path
   string sClassPath = sAppHome + "\\classes";
   string sOption_ClassPath =
      "-Djava.class.path=" + sClassPath;

   // setup VM options
   // vAddOption(string("-verbose"));
   vAddOption(sOption_ClassPath);
   vAddOption(sOption_AppHome);

   // initialize args
   JavaVMInitArgs vm_args;
   vm_args.version = 0x00010002;
   vm_args.options = vm_options;
   vm_args.nOptions = mctOptions;
   vm_args.ignoreUnrecognized = JNI_TRUE;

   // load jvm library
   HINSTANCE hJVM = LoadLibrary(sJVMpath.c_str());
   if( hJVM == NULL ){
      vShowLastError("Failed to load JVM from "
        + sJVMpath);
      return 0;
   }

   // try to start 1.2/3/4 VM
   // uses handle above to locate entry point
   CreateJavaVM lpfnCreateJavaVM = (CreateJavaVM)
      GetProcAddress(hJVM, "JNI_CreateJavaVM");
   jintVMStartupReturnValue = (*lpfnCreateJavaVM)
      (&jvm, &env, &vm_args);

   // test for success
   if (jintVMStartupReturnValue < 0) {
      string sErrorMessage = "Unable to create VM.";
      vShowError(sErrorMessage);
      vDestroyVM(env, jvm);
      return 0;
   }

   // find startup class
   string sStartupClass = "javabunny/JavaBunny";
   // notice dots are translated to slashes
   jclassStartup =
      env->FindClass(sStartupClass.c_str());
   if (jclassStartup == NULL) {
      string sErrorMessage =
        "Unable to find startup class [" +
        sStartupClass + "]";
      vShowError(sErrorMessage);
      vDestroyVM(env, jvm);
      return 0;
   }

   // find startup method
   string sStartupMethod_Identifier = "main";
   string sStartupMethod_TypeDescriptor =
     "([Ljava/lang/String;)V";
   midStartup =
     env->GetStaticMethodID(jclassStartup,
      sStartupMethod_Identifier.c_str(),
```

nsoftware

www.nsoftware.com

```
      sStartupMethod_TypeDescriptor.c_str());
   if (midStartup == NULL) {
      string sErrorMessage =
      "Unable to find startup method ["
      + sStartupClass + "."
      + sStartupMethod_Identifier
      + "] with type descriptor [" +
      sStartupMethod_TypeDescriptor + "]";
      vShowError(sErrorMessage);
      vDestroyVM(env, jvm);
      return 0;
   }

   // create array of args to startup method
   jstring jstringExampleArg;
   jclass jclassString;
   jobjectArray jobjectArray_args;
   jstringExampleArg =
     env->NewStringUTF("example string");
   if (jstringExampleArg == NULL){
      vDestroyVM(env, jvm);
      return 0;
   }
   jclassString =
     env->FindClass("java/lang/String");
   jobjectArray_args =
     env->NewObjectArray(1, jclassString,
     jstringExampleArg);
   if (jobjectArray_args == NULL){
      vDestroyVM(env, jvm);
 return 0;
   }

   // call the startup method -
   // this starts the Java program
   env->CallStaticVoidMethod(jclassStartup,
     midStartup, jobjectArray_args);

   // attempt to detach main thread before exiting
   if (jvm->DetachCurrentThread() != 0) {
      vShowError("Could not detach main thread.\n");
   }

   // this call will hang as long as there are
   // non-daemon threads remaining
   jvm->DestroyJavaVM();

   return 0;

}

void vDestroyVM(JNIEnv *env, JavaVM *jvm)
{
   if (env->ExceptionOccurred()) {
      env->ExceptionDescribe();
   }
   jvm->DestroyJavaVM();
}

void vShowError(string sError) {
   MessageBox(NULL, sError.c_str(),
     "Model App Error", MB_OK);
}

/* Shows an error message in an OK box with the
   system GetLastError appended in brackets */
void vShowLastError(string sLocalError) {
   LPVOID lpSystemMsgBuf;
   FormatMessage(
      FORMAT_MESSAGE_ALLOCATE_BUFFER |
      FORMAT_MESSAGE_FROM_SYSTEM |
      FORMAT_MESSAGE_IGNORE_INSERTS,
 NULL,
      GetLastError(),
 MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
      (LPTSTR) &lpSystemMsgBuf,    0,    NULL );
      string sSystemError =
        string((LPTSTR)lpSystemMsgBuf);
 vShowError(sLocalError +
        " [" + sSystemError + "]");
}

void vAddOption(string& sValue) {
   mctOptions++;
   if (mctOptions >= mctOptionCapacity) {
      if (mctOptionCapacity == 0) {
         mctOptionCapacity = 3;
         vm_options =
(JavaVMOption*)malloc(mctOptionCapacity *
  sizeof(JavaVMOption));
 } else {
         JavaVMOption *tmp;
         mctOptionCapacity *= 2;
         tmp =
(JavaVMOption*)malloc(mctOptionCapacity *
  sizeof(JavaVMOption));
         memcpy(tmp, vm_options, (mctOptions-1) *
  sizeof(JavaVMOption));
         free(vm_options);
         vm_options = tmp;
      }
   }
   vm_options[mctOptions-1].optionString =
  (char*)sValue.c_str();
}

/* If buffer is "c:\app\bin\java",
 * then put "c:\app" into buf. */
jboolean GetApplicationHome(char *buf, jint sz) {
   char *cp;
   GetModuleFileName(0, buf, sz);
   *strrchr(buf, '\\') = '\0';
   if ((cp = strrchr(buf, '\\')) == 0) {
      // This happens if the application is in a
 // drive root, and there is no bin directory.
      buf[0] = '\0';
      return JNI_FALSE;
   }
   return JNI_TRUE;
}
```

Download the Code!
www.JavaDevelopersJournal.com

int

www.int.com

actuate

www.actuate.com

JASON R. BRIGGS J2ME EDITOR

The Sky Is Falling

Apparently it hasn't been a good quarter for many PDA makers. Shipments were down from the same period last year so, of course, doom and gloom are predicted by all and sundry. Actually I'm exaggerating; one of the reports I read was fairly evenhanded in its approach – another was about as subdued as Chicken Little.

Interesting timing, then, for aJile Systems to announce a new 100% Java "proof-of-concept" PDA that utilizes their aJ-100 processor. The press release is typically effusive in the way that only press releases can be:

The result is a 10X performance enhancement that delivers full-motion, 16-bit color animation on the wireless mobile device's 320x240 QVGA display – performance that rivals the Java execution of a desktop PC...

However, perhaps this isn't just interesting timing on aJile's part – perhaps it shows a little strategic forethought. There's a fairly well-known piece of "market wisdom" that the best time to invest in R&D is in a downturn, so your company is ready for the inevitable upward swing in the market.

Maybe the same is also true for garnering interest in a product like aJile's PDA reference platform – the best time to do it is when everyone is moaning and groaning that the sky is falling and it's the end of the (PDA) world.

Of course, in the case of this particular Java PDA, there are bound to be arguments over the relative merits of Java-on-the-processor, as opposed to Java coprocessors, as opposed to Monty-style VMs, etc., etc., as the best way to go. However, I'm a firm believer in "the more the merrier." It seems to me that it doesn't matter in the long run which approach you use to run your Java code – as long as you're running Java (a standing-on-the-fence attitude which I expect will win me no fans).

I'm keen to see if the 10x performance enhancement, claimed by aJile, actually carries through if/when a manufacturer licenses the technology and builds a product on the back of it. (Although, considering the fact that Sharp in NZ still has no plans to distribute the Zaurus over here, it's entirely likely that by the time I see an actual product based on aJile's design in New Zealand, I'll be too old and senile to care.)

I've recently noticed that there have been a number of J2ME specification requests added this year, including:

- ***JSR 169:*** JDBC optional package for CDC/foundation profile
- ***JSR 172:*** J2ME Web services specification
- ***JSR 177:*** Security and trust services API for J2ME
- ***JSR 179:*** Location API for J2ME
- ***JSR 180:*** SIP API for J2ME
- ***JSR 184:*** Mobile 3D graphics API for J2ME
- ***JSR 190:*** Event-tracking API for J2ME

It's an impressive group. I also noticed that there was one rejection in the newly proposed J2ME APIs: the mobile game API. Considering that some of the technology a mobile game API could provide will be covered by a number of other JSRs already in development, this fills me with a little more confidence that the JCP is not accepting every proposal that comes in as a matter of course.

While all these APIs aren't necessarily targeted at the same group of devices, I do have one worry that diversification of J2ME APIs will potentially make life rather difficult in the future. Not just for developers, who have to navigate a virtual minefield of JSRs, but for the end users: do you buy phone A, which supports mobile media; phone B, which supports mobile 3D; or phone C, which supports both but is twice as expensive? I predict that OTA (over-the-air) download will be an interesting problem in compatibility checking in the not-too-distant future.

jasonbriggs@sys-con.com

AUTHOR BIO

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

qualcomm

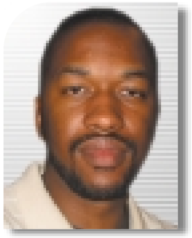
www.qualcom.com

Wireless J2ME Applications with Java and Bluetooth

Share and collaborate in a wireless network

Part 2 of 2

WRITTEN BY BRUCE HOPKINS



This article is the second installment in a two-part series on Java and Bluetooth. Last month you got your feet wet in Bluetooth (*JDJ*, Vol. 7, issue 8); if you don't remember what the role of a Bluetooth stack or a Bluetooth profile is in the context of a Bluetooth application, refer to Part 1.

Let's recap. The basic concepts of any Bluetooth application (Java or otherwise) consist of the following components:

- Stack initialization
- Device management
- Device discovery
- Service discovery
- Service registration
- Communication

We covered stack initialization last month, so let's proceed!

Device Management

LocalDevice and RemoteDevice are the two main classes in the Java Bluetooth Specification that let you perform device management. These classes allow you to request some statistical information about your Bluetooth device (LocalDevice) as well as some information about the devices in the area (RemoteDevice). The static method LocalDevice.getLocalDevice() returns an instantiated LocalDevice object for you to use. To get the unique address of your Bluetooth radio, call getBluetoothAddress() on your local device object. The Bluetooth address serves the same purpose as the MAC address on the network card of your computer; every Bluetooth device has a unique address. If you want other Bluetooth devices in the area to find you, call the setDiscoverable() method in LocalDevice object.

That's about all it takes to perform device management with the Java Bluetooth Specification APIs. Now, let's look at the concept that allows you to discover other Bluetooth devices: device discovery.

Device Discovery

Your Bluetooth device doesn't know what other Bluetooth devices are in the area. Perhaps there are laptops, desktops, printers, mobile phones, or PDAs nearby. Who knows? The possibilities are endless. To find out, your Bluetooth device uses Device Discovery classes that are provided in the Java Bluetooth API to see what else is out there.

The two classes required for your Bluetooth device to discover remote Bluetooth devices in the area are DiscoveryAgent and DiscoveryListener.

After getting a LocalDevice object, instantiate a DiscoveryAgent by calling LocalDevice.getDiscoveryAgent().

```
LocalDevice localdevice =
 LocalDevice.getLocalDevice();
DiscoveryAgent discoveryagent =
 localdevice.getDiscoveryAgent();
```

The are multiple ways to discover remote Bluetooth devices, but I'll discuss one particular method. First, your object must implement the DiscoveryListener interface. This interface works like any listener, so it'll notify you when an event occurs. In this case, you'll be notified when Bluetooth devices are in the area. To start the discovery process, call the startInquiry() method on your DiscoveryAgent. This method is nonblocking, so you're free to do other things while you wait for other Bluetooth devices to be found.

When a Bluetooth device is found, the JVM calls the deviceDiscovered() method of the class that implemented the DiscoveryListener interface. This method passes you a RemoteDevice object that represents the device discovered by the inquiry.

Service Discovery

Now that you know how to find other Bluetooth devices, it would be nice to see what services they offer. Of course, if the RemoteDevice is a printer, you know it can offer a printing service. But what if the RemoteDevice is a computer? Would it readily come to mind that you can also print to a printer server?

That's where service discovery comes in. You can never be sure what services a RemoteDevice may offer, so service discovery helps you find out.

Service discovery is just like device discovery – you use the DiscoveryAgent to do the "discovering." The searchServices() method of the DiscoveryAgent class allows you to search for services on a RemoteDevice. When services are found, the servicesDiscovered() method will be called by the JVM if your object implemented the DiscoveryListener interface. This callback method also passes in a ServiceRecord object that pertains to the service you searched for. With a ServiceRecord in hand, you can do plenty of things, but you would most likely want to connect to the RemoteDevice where this ServiceRecord originated:

```
String connectionURL =
 servRecord[i].getConnectionURL(0,
 false);
```

Service Registration

Before a Bluetooth client device can use service discovery on a Bluetooth server device, the Bluetooth server needs to register its services internally in the service discovery database (SDDB). This process is called service registration.

Note: In a peer-to-peer application, such as a file transfer or chat application, any device can act as the client or the server, so you'll need to incorporate that functionality (both client and server) into your code in order to handle both scenarios of service discovery (the client) and service registration (the server).

Here's an example of what's involved in getting your service registered and stored in the SDDB (Listing 1 provides the code; which can be downloaded from www.sys-con/java/sourcec.cfm):

1. Call Connector.open() and cast the resulting connection to a Stream-

hit software

www.hitsw.com

ConnectionNotifier. Connector.open() creates a new ServiceRecord and sets some attributes.

2. Use the LocalDevice object and the StreamConnectionNotifier to obtain the ServiceRecord that was created by the system.
3. Add or modify the attributes in the ServiceRecord (optional).
4. Use the StreamConnectionNotifier and call acceptAndOpen() and wait for Bluetooth clients to discover this service and connect. The system creates a service record in the SDDB.
5. Wait until a client connects.
6. When the server is ready to exit, call close() on the StreamConnectionNotifier. The system removes the service record from the SDDB.

StreamConnectionNotifier and Connector come from the javax.microedition.io package of the J2ME platform.

That's all that you need to do service registration in Bluetooth. The next step is communication.

Communication

Okay, Bluetooth is a communication protocol, so how do you communicate with it? Well, the Java Bluetooth API gives you three ways to send and receive data, but for now we'll cover only one of them, RFCOMM. RFCOMM is the protocol layer that the Serial Port Profile (SPP) uses to communicate, but these two items are almost always used synonymously.

Server Connections with the Serial Port Profile

Listing 2 demonstrates how to open a connection on a Bluetooth device that will act as a server. For the most part, this is the same code used in service registration; service registration and server communication are both accomplished using the same lines of code.

Here are a few items that I want to point out: the string URL begins with btspp://localhost, which is required if you're going to use the Bluetooth Serial Port Profile. Next comes the UUID part of the URL, which is 0011223344556677 889900AABBCCDDEEFF. This is simply a custom UUID that I made up for this service; I could have chosen any string that was either 32-bits or 128-bits long. Finally, we have ;name=serialconn in the URL string. I could have left this part off, but I want my custom service to have a name, so the actual service record in the SDDB has the following entry:

```
ServiceName = serialconn
```

The implementation also assigned a channel identifier to this service. The client must provide the channel number along with other parameters in order to connect to a server.

Client Connections with the Serial Port Profile

Establishing a connection with the SPP for a J2ME client is simple because the paradigm hasn't changed for J2ME I/O. Simply call Connector.open().

```
StreamConnection con =

(StreamConnection)Connector.open(url);
```

Obtain the URL string that you need to connect to the device from the ServiceRecord object you get from service discovery. The following is a more complete demonstration of how an SPP client makes a connection to an SPP server.

```
String connectionURL =
 serviceRecord.getConnectionURL(0,
 false);
StreamConnection con =
 (StreamConnection)Connector.open(con-
  nectionURL);
```

What does an SPP client connection URL look like? If the address of the server is 0001234567AB, the string for the SPP client would look something like this:

```
btspp://0001234567AB:3
```

The "3" at the end of the URL string is the channel number that the server assigned to this service when it was added to the SDDB.

Java Bluetooth Development Kits

Who makes this stuff and how can you get your hands on it? Here are your options:

The JSR-82 Reference Implementation was created by Motorola, so they're responsible for distributing it (www.motorola.com/java). Unfortunately, this Reference Implementation is only the JSR-82 APIs (see Figure 1, section B) so you won't be able to do much with it even if you have some Bluetooth hardware. To get the JSR-82 Reference Implementation to work on your development system, you need to license the Motorola CLDC as well.

If you already have Bluetooth devices for your development computers, try out the JSR-82–compliant Java Bluetooth solution from Atinav (www.atinav.com). They support numerous RS-232, UART, USB, CF, and PCMCIA Bluetooth devices. Their solution is based on an all-Java stack and their SDK includes the following profiles: GAP, SDAP, SPP, and GOEP. This solution also includes a KVM, so you only need to obtain the Bluetooth hardware to create and test your apps.

Rococo (www.rococosoft.com) is most famous for their Java Bluetooth simulator, although they also make a Java Bluetooth developer kit for the Palm OS. Using the Impronto Simulator, Java developers only need a single computer to compile and test their Java Bluetooth applications. The simulator supports GAP, SDAP, SPP, and GOEP profiles, and is currently priced at $1,000.

Esmertec (www.esmertec.com) is well known for their Jbed Micro Edition CLDC implementation and the Jbed RTOS package. The Jbed platform is based upon their FastBCC technology, which dynamically loads and executes Java bytecode at native speeds (much like the goals of the Project Monty KVMs). One key advantage of Esmertec is that the Jbed CLDC implementation can run on a variety of different hardware platforms – with or without an operating system! Esmertec implemented their Bluetooth stack completely in Java, and they support GAP, SDAP, SPP, LAP, and PAN on a wide array of Bluetooth hardware modules. They also included support for Palm OS and Pocket PC devices.

Smart Network Devices (www.smart-nd.com) has a unique approach in their Java Bluetooth development kit; they include all the components shown in Figure 1, section B, for a complete Java Bluetooth–enabled device: a Bluetooth device (radio), a Bluetooth stack, Bluetooth profiles, a KVM, and the JSR-82 Java Bluetooth APIs. Their product is called Micro BlueTarget Starter Kit – Java version (see Figure 2), based on the Micro BlueTarget reference board. It supports the GAP, SDAP, SPP, and GOEP profiles. The Micro BlueTarget is a not a peripheral (like

AUTHOR BIO
Bruce Hopkins is a senior Java consultant at Great Lakes Technologies Group in Southfield, MI. He has worked with Java for over six years, and has researched in wireless networking for four. Bruce is the coauthor of an upcoming book entitled Java Bluetooth by Apress (November 2002).

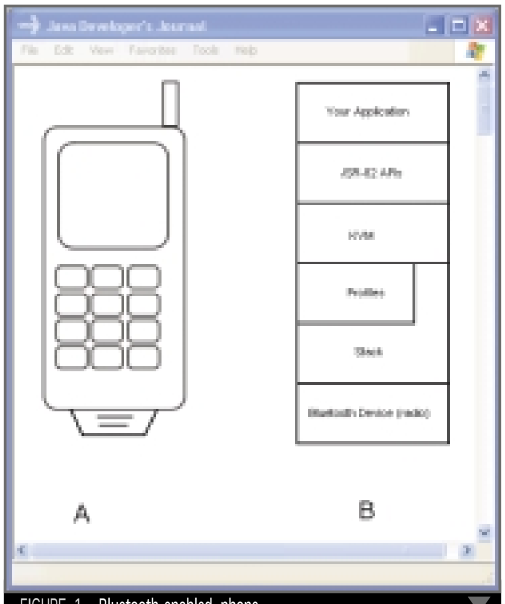


FIGURE 1 Bluetooth-enabled phone

sitraka

www.sitraka.com

the 3Com Bluetooth module in Figure 3). It's a self-contained Java Bluetooth host device, so it doesn't need to be connected to a host computer to operate. It has its own operating system and KVM built in, and you can download your code to the device using its Ethernet or RS232 interface. You can buy the complete Java Bluetooth development kit for $2,880 and the Micro BlueTarget boards in single quantities for $425.

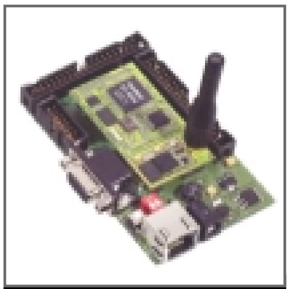


FIGURE 2 Micro BlueTarget Starter Kit – Java version

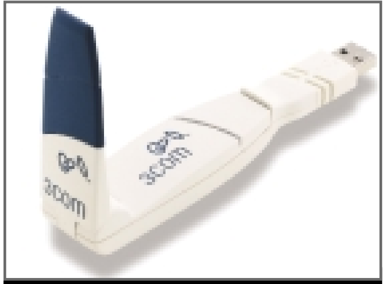


FIGURE 3 Bluetooth radio

The Advantages of Java and Bluetooth

Let's look at a scenario where life is made simpler using Java and Bluetooth technology: the Java Shared Whiteboard. Three employees of Acme Widgets Inc. need to have an impromptu meeting. Unfortunately, no conference rooms are available, so the team is forced to hold their meeting in the cafeteria. They would have preferred using a conference room because each room is equipped with an electronic whiteboard. However, since every member of the team has a Java Bluetooth–enabled PDA, their meeting in the cafeteria is very productive.

One member has a new program for his PDA called the Java Shared Whiteboard. Using Bluetooth technology, he sends that program to the rest of the team. Using Over-the-Air Provisioning (OTA) provided by J2ME, each member installs and runs the application on the fly. The meeting can now begin because the whiteboard is shared among the PDAs. Each participant can draw figures on his or her device and the image will appear instantly on the other screens. To save time, one member can take notes and send them to everyone's device while the meeting is in progress.

What are the benefits of Java and Bluetooth? Of course, Java gives you platform independence for the Shared Whiteboard application. Therefore, you don't need to worry about what kind of PDA your fellow team members have (Palm, Pocket PC, Sharp, Sony, Handspring, etc.) as long as a compatible KVM and libraries are available. Bluetooth has an additional benefit – it enables you to create instant wireless networks with these devices in order to collaborate and share data. This network is portable, so you can move it to the cafeteria, a conference room, or outside – it doesn't matter.

Summary

It's a great time to be a wireless developer. Bluetooth enables you to share and collaborate in ways never imagined. Now, I can enable my clunky old desktops, laptops, and PDAs to participate in a wireless network by simply adding a KVM and a Bluetooth radio. The fun is just beginning.

• • •

Book Overview

We've only scratched the surface with Bluetooth. In our upcoming book, *Java Bluetooth*, Ranjith Antony (my coauthor) and I show how to use Java and Bluetooth with multiple vendor SDKs and Bluetooth devices. We'll also cover many practical scenarios for using Java and Bluetooth in the real world for file transfer, security, and encryption using a simulator, wireless printing, and much more. The advanced chapters will even show you how to use Bluetooth in a Jini network.

javaspaces@comcast.net

sys-con media

fiorano

www.fiorano.com

Whole House Audio from the Palm of Your Hand

Working out the quirks

Part 2 of 3

written by Bill Ray

In Part 1 of this series (JDJ, Vol. 7, issue 6), I showed how I developed an MP3 player in Java, and then added the ability to control that player from a wireless handheld device using a PersonalJava application.

While I could only stop, pause, adjust the volume, and select the next track to be played, I still found the application useful, but not yet perfect…

The first problem to be addressed was the combinations you can get when listening to your entire music collection at random. When a nice relaxing Enya track fades out and you find yourself launching into the world of Eminem, the shock can be considerable, not to mention that sometimes I'm just not in the mood for Aztec Camera…but have no preference beyond that. So, some sort of weighting is needed.

My next problem was the networking side of things. Wireless Ethernet is nice, but not ideal on the iPaq. In addition to the devastating impact it has on the battery life, I can't get it to connect automatically on demand. This is because I use a GPRS phone for my mobile Internet access (thus the method of connectivity must be specified) and the networking interface on Pocket PC devices isn't very good. So when I'm using the application, my iPaq has to be on all the time with the drain of networking, or I have to manually connect to the network whenever I want to control the music. Neither option is useful, and as my iPaq has Bluetooth built-in (it's a 3870), this should provide an ideal wireless mechanism.

Listening Preferences

Starting with the weighting, I decided that each track needed a descriptive profile. Since I don't want to type these in when I rip the tracks, there need to be default values that can be adjusted later. I want to be able to adjust the settings of the current track from the handheld, so if I'm listening to a track and decide I like it a lot, I can increase the chances of it playing again, or if I don't like it, downgrade it so it's unlikely to pop up later.

After some consideration I came up with a few criteria. Each track would have the following fields, rated from 1 to 100: how much I like the track, how fast it is, how loud, how instrumental, and the name of the track that should follow. The last field was put in as I have some tracks that should really be put together to make sense. The fast and loud fields refer to how energizing and rousing I feel the music is – I like loud music in the morning, fast music when I'm sewing, and instrumental music when I'm programming. With this in mind the user must be able to adjust his or her current listening preferences, based on the same fields (though not "liked," obviously).

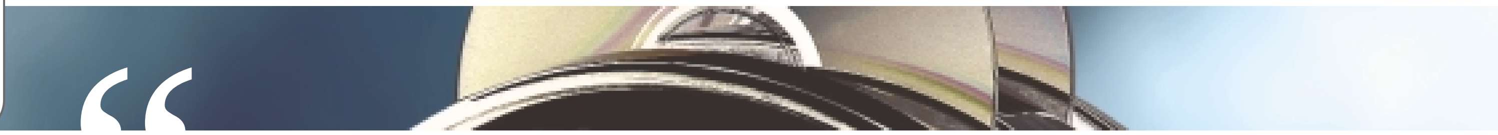
Therefore, the process of deciding on the next track will be as follows: once a track starts playing, the track specified in the "preferred next track" field will be located and passed the current listening preferences to see if it matches them. If it does, it will be played next. If it doesn't match well enough (with a random element), another track will be selected at random and asked if it matches the preferences. By default the preferred next track is the following track on the album, though the user can change this along with the other preferences. This way several tracks can be selected until one that matches what the user wants to hear is found.

Note that a random element is also introduced, so even tracks that are not liked much can pop up on occasion. Of course, if I really hated them, I could always delete the files. I also decided that it made sense to reduce the "liked" field by one point each time the track was played, and increase it by a point each time the track was rejected. This way my collection should, in theory, rotate gently and ensure I get to hear everything over time. After some discussion, I also included a Boolean field called "aletiaApproved". This notes if the track is liked by my wife and allows me to specify that only such tracks should be played. My music tastes are fairly broad and not shared by everyone, so this also provides a general liked-by-most-people criterion.

Originally, I thought I would just serialize the Track objects into a file and read them back, but this was extremely optimistic. The Track objects contain media players and all sorts of objects that can't be serialized. I ended up creating a new class just to contain the track details; this can be constructed from a Track object or by specifying the field values as parameters. This means that when I wish to save the values (when the application is shut down), I need to create an ArrayList of TrackDetails objects with an entry for each track, and then serialize this to the local disk.

This worked fine, but loading was more complex as I not only had to load the file and create the Track objects from it, I also had to add any new tracks that had been created and remove any that had been deleted. I decided to store the details in the same directory that was selected for playback; this way I could have multiple files containing track details without worrying about which one was being used.

The application therefore looks in the current directory to see if such a file exists. If so, it loads it, then goes through its normal scanning process. Before adding any MP3 files to the main playlist, it checks to see if it already has details for that track (by comparing the full file name and path) – if it does, it uses the loaded details, if not, it creates a default set. The track is then added to the main ArrayList. This enabled me to automatically add new tracks as well as remove tracks that had been deleted since the last time the application was run, since the loaded details are only copied to the live playlist when the file is located – tracks for which there are no files are automatically removed.



“I did this so my Psion Digital Radio can save recordings of documentaries and plays, which are not really suitable for randomly mixing with my music but should still be selectable on demand”

I also added a directory to my MP3 collection called NoRandomPlay. MP3 files in this directory would never be played at random, but could still be selected. I did this so my Psion Digital Radio can save recordings of documentaries and plays, which are not really suitable for randomly mixing with my music but should still be selectable on demand. Ideally, I'd like to be able to control my digital radio from my mobile, but that will have to wait for another series of articles.

Network Protocol

Having decided how the application was going to work, it was time to decide how to extend the network protocol to add this functionality. Again, it's always best to start with the server so it can easily be tested using HyperTerminal or something similar. It's clear that I'll need four commands in total, two for getting and setting the current listening preferences and two for getting and setting the preferences for a particular track. Given the noncritical nature of the application, I'm not too worried about the bandwidth used, and I'm making the whole thing case insensitive, as I did before:

```
[client]  Get Preferences
[server] +Preferences Follow
[server] Loud: 50
[server] Fast: 50
[server] Instrumental: 50
[server] Aletia Approval: true
[client] Set Preferences:50#50*50-true
[server] +Preferences set
```

These commands will be used for getting and setting the current listening preferences. The latter of these deserves some explanation: the setting is done in a single line, with the preferences being specified as loud, fast, instrumental, and “Aletia approved”, with different dividers marking the space between them. Using different markers makes it harder for a human to read, but much easier to parse for the application; for example, to extract the loud field from an incoming preference the following line can be used:

```
int loud = new
Integer(message.substring(message.indexOf(',') + 1, mes
 sage.indexOf('#'))).intValue();
```

Basically this is laziness on my part. It would have been better to have a properly human-readable protocol both ways, but I didn't bother. Setting and getting the preferences for a particular track is almost identical:

```
[client] Style 3
[server] +Style Follows
[server] Number: 3
[server] Liked: 30
[server] Loud: 50
[server] Fast: 50
[server] Instrumental: 50
[server] Aletia Approved: true
[server] Next: 4
[client] Set_Style 3:30,50#50*50-true=12
+Style set
```

One additional field is specified: how much the track is liked. The number of the track is also returned; this is where the track lies in the ArrayList on the server and was included in case I ever wanted to set the style based on the file name, though that hasn't happened yet.

After I got the server working (so few words, so much work!) and tested with HyperTerminal, I started thinking about how to manage the user interface. (If this had been a commercial application, I would have probably started with the interface, but as the whole thing is intended only for my use, it was left until the end.) PersonalJava can support only one frame and one dialog, but that applies only to those concurrently visible. I'm already using one dialog for my track list-

inetsoft

www.inetsoft.com

ing, so I created another two to set listening preferences and set the preferences for a track. While I probably could have used one class for all the preference settings, I decided they were sufficiently different to warrant their own classes.

I really wanted a slider to set the various levels, but there's no Swing in PersonalJava. If I wanted one I would have to write one, which I wasn't keen on doing. Searching around among my old work I discovered a slider object I wrote years ago, before Swing existed, which suited me completely. As the class was so old, it was able to work with PersonalJava and looked considerably better than what I would have created this time around (see Figure 1). It's always nice to reuse a component, especially one you can't remember creating.

With my sliders it was fairly easy to put together the interface for setting track and listening preferences. When the user wants to set preferences, the client asks the server for either the current listening preferences or the track preferences as requested by the user, then sets the levels of the sliders to the right place and makes the dialog visible. Changes are written back to the server when the dialog is dismissed.

The track preferences dialog didn't have the space for me to allow the user to alter the preferred next track, so that will have to wait until the next version.

Overall the system works pretty well, generally playing two or three tracks from an album before moving on to the next one, which makes for a much better listening experience. Setting the preferences has an influence, but still allows for occasional surprises, though for some strange reason the application still has a thing about "Hey Mr. President" by 4 Non Blondes.

Networking

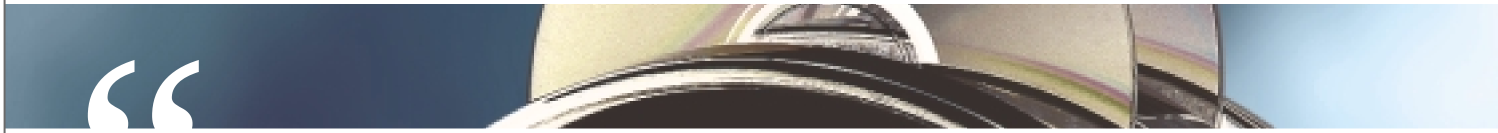
The next problem was the networking and required a more radical solution. I liked having multiple clients, often running the client on my PC when I was working and sending the instructions over the wired LAN, so that would have to stay, but something had to be done about the mobile devices.

Bluetooth provides the ideal mechanism for such an application, with low power consumption and the ability to connect very quickly. While the iPaq is Bluetooth-capable, it needs another Bluetooth device to speak to, so I invested in a Bluetooth Network Access Point from Inventel (I did actually need it for another project too; I'm not quite that frivolous). Given the relative youth of the protocol and my urgent need (for my other project), this arrived without a manual of any sort and with drivers for another device entirely. But with the helpful support of the Inventel people I managed to get it working. It turned out to be running Linux, which proved immensely helpful in getting it working.

With a device like this I could attach my iPaq to the network and route TCP/IP packets over my LAN, or even the Internet (though I still don't have DNS working properly) via my ADSL line (see Figure 2). I have to admit I was surprised when my application ran without modification, though I shouldn't have been. While it helped, the battery consumption was still too high and I had to connect manually each time.

It's also wrong. While connecting to a LAN is useful, it's not what I wanted to do. My application wants to connect to a server; the LAN is used only as an intermediate step and is redundant. Bluetooth is about connecting devices to each other, not to networks (Wireless Ethernet is perfectly good for these kind of applications), so I should try to connect directly to the server. In that case, the server will also need Bluetooth connectivity, which was provided by a TDK Bluetooth USB Adapter (with the added benefit of allowing both my iPaq and Palm handhelds to synchronize wirelessly). Now that I had the hardware in place, it was time to think about how to get the software working with it.

One of the problems with Java, in fact, the primary problem, is its inability to support hardware it hasn't heard of. While there's a JSR for Bluetooth, it's still a work-in-progress, and it will be a while before we see any implementation, so we're on our own for the moment. However, Bluetooth was designed with compatibility in mind, and within the protocol are various "profiles" that define how different applications might use Bluetooth without getting involved with the whole radio thing. One of these profiles defines a standard for serial communications, so it should be possible to create a serial connection from the client to the server from Java. The iPaq defines two serial ports as working with Bluetooth: COM8 is used for outgoing connections while COM7 is used for incoming. On the PC these change to COM3 and COM4, respectively, though configurable through the TDK software. Of course, when you open a serial connection you don't normally specify a destination, so some experimentation was necessary to establish where an outgoing connection would end up. This



"Bluetooth was designed with compatibility in mind, and within the protocol are various 'profiles' that define how different applications might use Bluetooth without getting involved with the whole radio thing"

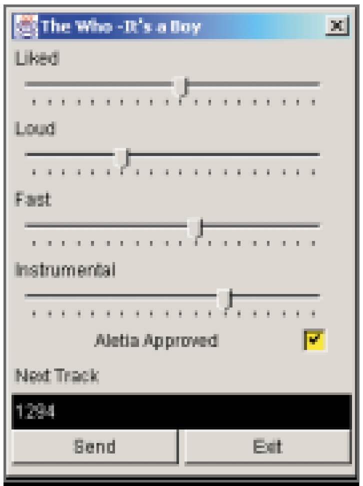


FIGURE 1 Slider

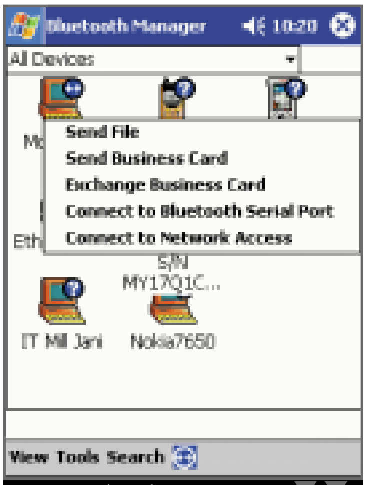


FIGURE 2 Bluetooth Manager

improv tech

improv-tech.com

showed an outgoing serial connection will connect to the last place a serial connection was made to, so getting it working was simply a matter of making the connection manually once, then running the software when required.

While this sounds good, there's one problem: serial connections are not part of PersonalJava. Not all PersonalJava devices are considered to have a serial port so we're reduced to the lowest common denominator. Luckily James Nord fixed this particular failing with a serial API that works on an iPaq and other Pocket PC devices, and is available free from his Web site (see Useful Links at the end of this article). He's never actually used it with Bluetooth, but helpfully pointed out that I was using an old version that didn't support the event-driven reception of data; it works perfectly with the latest version.

Testing the connection was relatively easy once I got the right version of the Serial API on my iPaq and the server; I used the same code on both. Opening the port is a matter of working through the available ports until one matches the desired port:

```
String port = "COM3";
portList = CommPortIdentifier.getPortIdentifiers();
while (portList.hasMoreElements()) {
  portId = (CommPortIdentifier) portList.nextElement();
  if (portId.getPortType() == CommPortIdentifier.PORT_SER-
IAL) {
    if (portId.getName().startsWith(port)) {
      CreateConnection();
    }
  }
}
```

Once the port has been identified, it's opened in the normal way:

```
serialPort = (SerialPort) portId.open("Test3", 2000);
inputStream = serialPort.getInputStream();
outputStream = serialPort.getOutputStream();
serialPort.addEventListener(this);
serialPort.notifyOnDataAvailable(true);
```

Setting the parameters is optional, since it won't matter if they don't match. Any parameters set will be used only in the connection between the Java application and the Bluetooth stack, but they can be set for completeness:

```
serialPort.setSerialPortParams(57600, SerialPort.DATA
 BITS_8, SerialPort.STOPBITS_1, SerialPort.PARITY_NONE);
```

From this point it's just a matter of writing data as required and reading it back through events:

```
outputStream.write("My Message".getBytes());
outputStream.flush();
```

Remember to flush the buffer. The availability of data is indicated through a normal event, so the data can be read in (see Listing 1).

"While this sounds good, there's one problem: serial connections are not part of PersonalJava"

Now the client and server can communicate over the serial link, which is established within a second or two of the application being launched. The same protocols can be used, and the server simply needs to launch a second thread for listening and reporting serial communications (actually, it doesn't need to be a thread as the serial reading is not a blocking action). As the server was designed to cope with multiple clients, it has no problems with this and the application works.

Conclusion

Now I'm getting there: my application plays the music I like, tailored to fit my mood, and I can control it by simply turning on my iPaq and running the application; but the project isn't over yet. Some time ago I wired the whole house with speakers, as I like music wherever I go, but banks of switches are not ideal; more than once I've wandered upstairs to find that what we're watching on TV is being blasted in the bedroom. Now that I can control my music from my mobile, I need to start controlling the sound. In Part 3 I'll be adding the ability to turn speakers on and off from my application.

Useful Links

- *Pocket PC Serial Library:* www.teilo.net/software
- *Desktop Serial Library:* http://java.sun.com/products/javacomm/index.html
- *Bluetooth Bits:* www.expansys.com
- *Digital Radio:* www.psion.com

Author Bio

Bill Ray has worked for several telecommunications companies around Europe, including Swisscom where he was responsible for the development of their Java-compatible DTV platform. He is security editor for Wireless Business & Technology *and coauthor of* Professional Mobile Java Development, *published by Wrox Press.*

bill@network23.co.uk

Listing 1

```
public void serialEvent(SerialPortEvent event) {
  if (event.getEventType() == SerialPortEvent.DATA_AVAILABLE) {
    byte[] readBuffer = new byte[inputStream.available];
    String message = "";
    try {
      while (inputStream.available() > 0) {
        int numBytes = inputStream.read(readBuffer);
      }
      message = new String(readBuffer);
      System.out.println("Received :" + message);
    } catch (IOException e) {
      e.printStackTrace();
    }
}
```

Download the Code! www.JavaDevelopersJournal.com

zerog

www.zerog.com

FULCRUM Professional Edition 1.1

by AccelTree

REVIEWED BY KEDAR GODSE *kedar@harbinger-systems.com*

info

AccelTree Software
4th Floor, KPCS House
Bhusari Colony, Paud Road
Kothrud, Pune 411 038
India
Web: www.acceltree.com
Phone: 91-20-528 5881
Fax: 91-20-528 5884
E-mail: info@acceltree.com

Specs
Platforms: Windows NT/2000/XP
Databases: Oracle, DB2
Pricing: $495

Test Platform
Intel Pentium IV, 20GB disk, 128MB of memory, Windows 2000, Oracle8*i* database, JRun Web server

Over the past couple of years, a number of Java development tools have appeared on the market; these tools focus on various aspects of software development, such as modeling, deployment, and testing, and aim to increase productivity. As a developer, prominent on my wish list is a productivity tool that addresses code development. The main criteria when looking for this tool was that it should assemble code quickly as well as be flexible enough for me to change the assembled code. AccelTree's FULCRUM promises all this and more.

FULCRUM speeds up core tasks – code assembly, data structure definition, validations, documentation, and more. The driving concept of FULCRUM is the generalization of repeated patterns of Java code in the form of templates that can be used as "building blocks" to construct efficient Java objects and applications.

Templates

The palette of templates that the product provides includes Java classes, EJBs, class methods, code blocks, and program specification templates that generate documentation automatically. Also included are predefined JavaScript front-end validations for data formats and numeric validations that can be linked to HTML controls using FULCRUM's Presentation Manager. While FULCRUM doesn't create the HTML file, it generates XML and XSL on the fly, which can then be rendered on a browser.

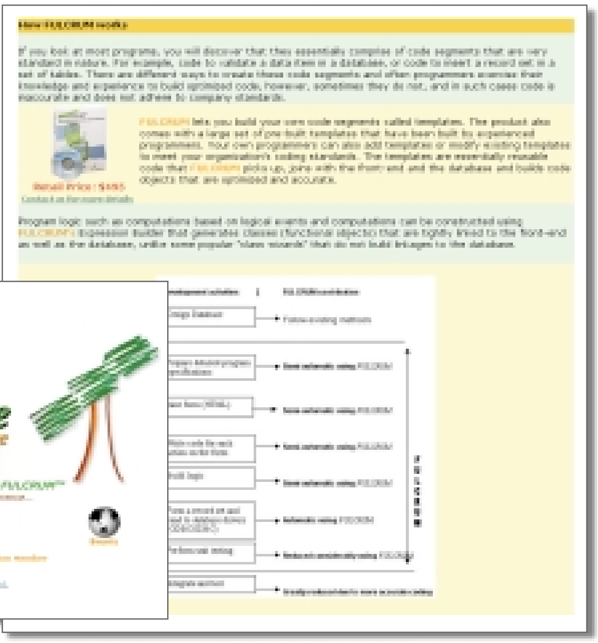
While most complex data structures supported by Java are meaningful only at runtime, FULCRUM provides a virtual configuration mechanism to help developers get the meaning by looking at the vectors or data structures configured within FULCRUM. This also reduces documentation needs. The product features a business rule engine to which the middleware components can make calls as needed at runtime. This satisfies a critical need of application configuration by enabling you to edit business rules without modifying code.

Installing and Using FULCRUM

Getting started using the installation CD for version 1.1 was a breeze. The requirements for FULCRUM include J2SDK version 1.3, ActiveX Bridge 1.0 (Java plug-in), Microsoft Windows Installer, Microsoft XML Parser 3.0, and MDAC 2.5. The setup wizard lets you include these in the course of installation if you don't have them already.

Java application development with FULCRUM does not require any additional runtime software. Setup was essentially smooth and in a couple of minutes I was checking out the product features. I had previously checked out version 1.0 and found that getting the hang of the development flow was not exactly a piece of cake. The documentation gave me a basic idea about the tool, but left me befuddled in terms of actually using the features. The online documentation was not very clear either. I needed better guidance with concrete examples to visualize how templates were used within FULCRUM to develop and test an application as well as to see how it could help me with code assembly.

Version 1.1 turned out to be a vast improvement, in this respect. The Help documentation provided an extensive introduction to the concepts and included a comprehensive FAQ. It also came with a tutorial guide and CBT, which I hoped would get me up and running in a couple of hours. Not quite. Though the tutorial examples were lucidly written, it took me almost two days to gain enough of a comfort level with the product features to really start using the tool.


How FULCRUM works

If you look at most programs, you will discover that they essentially comprise of code segments that are very standard in nature. For example, code to validate a data item in a database, or code to insert a record set in a set of tables. There are different ways to create these code segments and often programmers exercise their knowledge and experience to build optimized code, however, sometimes they do not, and in such cases code is inaccurate and does not adhere to company standards.

FULCRUM lets you build your core code segments called templates. The product also comes with a large set of pre-built templates that have been built by experienced programmers. Your own programmers can also add templates or modify existing templates to meet your organization's coding standards. The templates are essentially reusable code that FULCRUM picks up, joins with the front-end and the database and builds code objects that are optimized and accurate.

Retail Price : $495
Contact us for more details

Program logic such as computations based on logical events and computations can be constructed using FULCRUM's Expression Builder that generates classes (functional objects) that are tightly linked to the front-end as well as the database, unlike some popular "class wizards" that do not build linkages to the database.

Since we were in the process of designing an in-house resource management system, I decided to use FULCRUM to quickly develop a simple application framework and get a sense of how helpful the tool would be. The application specs required three Java classes for maintaining employee data, maintaining project data, and assigning appropriate employees to projects.

After briefly reviewing the tutorial and initiating the project within FULCRUM, I defined the program specifications through the FULCRUM Program Specification Wizard, which promptly generated a program specification document (see Figure 1). The wizard also helped me define the class, select the appropriate template from FULCRUM's template library, and set up the naming conventions for the class, methods, and variables. The defined class is saved as a .java file.

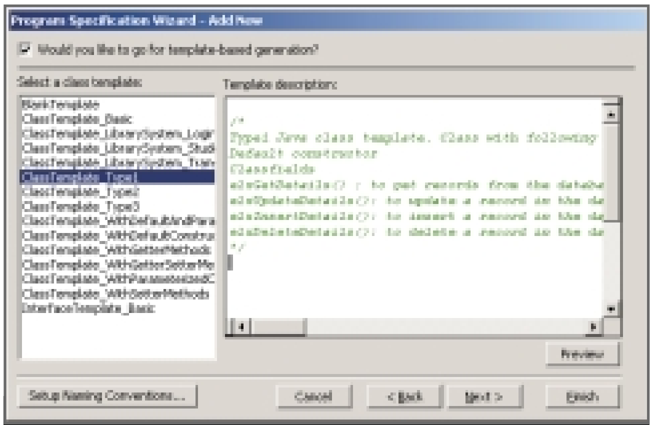
I included the Java file within the project module using the FULCRUM Project Explorer and added variables and methods to the class using the class builder functionality, which also provides an advanced code editor so you can enter your own code. The editor supports syntax highlighting for Java and HTML in addition to the usual editing features.

To test the class, FULCRUM generates a test JSP that passes dummy values as parameters to the methods and displays the returned data in a browser window. I used FULCRUM's Presentation Manager to map between HTML controls and Java class variables (see Figure 2), and FULCRUM generated the Presentation JSP based on this.

Although the module I've described here is quite simple, checking out all the FULCRUM features and implementing them took me several hours. However, the time invested helped me develop the other classes speedily in about half the time I would typically expect.

north woods

p/u


Program Specification Wizard - Add New
Would you like to go for template-based generation?
Select a class template:
Template description:
Preview
Setup Naming Conventions...
Cancel
< Back
Next >
Finish

BlankTemplate
ClassTemplate_Basic
ClassTemplate_LibrarySystem_Logi
ClassTemplate_LibrarySystem_Stud
ClassTemplate_LibrarySystem_Tran
ClassTemplate_Type1
ClassTemplate_Type2
ClassTemplate_Type3
ClassTemplate_WithDefaultAndPara
ClassTemplate_WithDefaultConstru
ClassTemplate_WithGetterMethods
ClassTemplate_WithGetterSetterMe
ClassTemplate_WithParameterizedC
ClassTemplate_WithSetterMethods
InterfaceTemplate_Basic

```
/*
Typed Java class template. Class with following
Default constructor
Classfields
[illegible]GetDetails() : to get records from the databa
[illegible]UpdateDetails(): to update a record in the da
[illegible]InsertDetails(): to insert a record in the da
[illegible]DeleteDetails(): to delete a record in the da
*/
```

FIGURE 1 Program Specification Builder for template selection

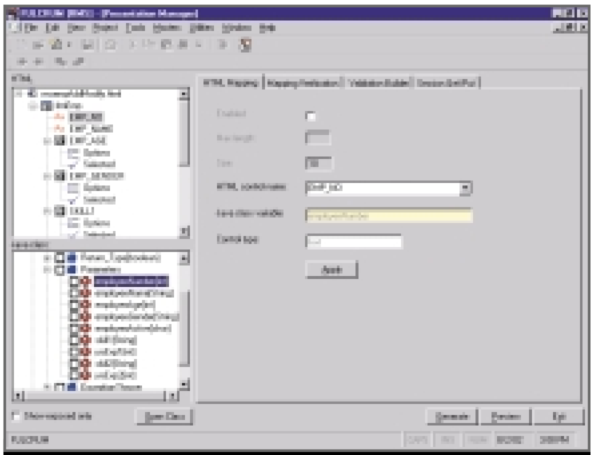


FIGURE 2 Mapping of HTML controls and Java class variables

Summary

Though the benefit of using templates for code development is intuitively obvious, the way FULCRUM integrates this concept across the development process from specification to testing gives the tool its real power and value. If reinventing the wheel is not your hobby and you need a tool to take care of the tedious development overhead, the FULCRUM Java code assembler is what you've been looking for. Be prepared to spend a couple of days learning its concepts, though it will be time well spent. While a skilled or expert Java programmer can utilize this tool to the maximum advantage in large developments, even a programmer with modest Java experience will gain insight into overall architecture by using the FULCRUM process for development.

JDJ Product Snapshot

Target Audience: Java developers, technical architects, application designers

Level: Advanced beginner to skilled

Pros:
- Speedy creation of classes and methods
- Quicker code assembly, checking, validation, and testing
- Product comes with CBT
- Well-written Help documentation and tutorial

Cons:
- Unlike an IDE, debugging and creation of HTML screens need to be done external to FULCRUM

it toolbox

www.ittoolbox.com

IntelliJ IDEA 3.0

by JetBrains, Inc.

REVIEWED BY DUANE FIELDS duane@deepmagic.com

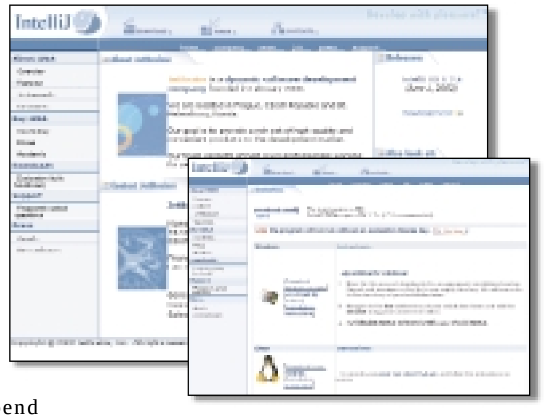
info

JetBrains, Inc.
(formerly IntelliJ Software)
Klanova 9/506
14700 Prague
Czech Republic
Web: www.intellij.com/jetbrains/
Phone: +7 (812) 380-1641
Fax: +7 (812) 380-1643
E-mail: info@intellij.com

Platforms
Windows, Mac, Linux, other JDK 1.3 platforms

Pricing
$395 (quantity discounts available)

Test Platforms
- AMD 1.33GHz, 784MB of RAM, Windows XP
- 700MHz Intel Pentium, 256MB of RAM, Windows 2000



I must admit, until recently my idea of an integrated development environment was Emacs, a couple of shell windows, and a six-pack of Dr. Pepper. I had nothing against IDEs, in fact I was all for them, I just couldn't find one that worked for me, instead of the other way around. Everything I tried either didn't format code the way I liked, required the entire development team to convert to it, didn't run my build scripts, wouldn't talk to my source code control system, or otherwise forced me to bend to its will. Maybe I'm too picky, but hey – I like to do things my way.

For the past several months, however, I've been developing almost exclusively with various beta builds of IntelliJ IDEA 3.0. It still has some bugs, of course, but this new IDE is so spectacular that even in its preproduction state I can't imagine coding without it. I introduced a few co-workers to the software to hear their opinions, knowing each of them already had established their own favorites – NetBeans, Forte, JBuilder, JRun Studio, and Visual J++. They're all using IDEA 3.0 now.

IntelliJ IDEA 3.0

JetBrains, Inc. (formerly IntelliJ Software), is based in Prague, Czech Republic, and St. Petersburg, Russia. They have released a number of successful Java development tools, most notably their Java IDE, IntelliJ IDEA. The latest version, 3.0, is scheduled for release this fall and will include some major enhancements, such as full JSP/EJB support, integration with Ant and JUnit, XML support, and a rich plug-in API for developing extensions. Like IDEA 2.5, IDEA 3.0 includes support for macros and code generation shortcuts, all dramatically improved. Of course, all the basics are there as well – a nice debugger, code completion, searching, replacing, code formatting, syntax/error highlighting, and source code control system integration. The popup code completion and JavaDoc hints are first rate, and super easy to use and configure. I'm finding new features every day and, unfortunately, it's impossible to cover all the goodies in one article.

Notably absent from the feature list are GUI builder tools like the ones found in NetBeans and other IDEs. While there's plenty of room for discussion regarding the vices and virtues of GUI builders, it's an important distinction worth noting for those who find them useful.

The IDE itself is written in Java and, unlike Eclipse, uses Swing for the GUI. While complex Swing applications tend to suffer from Java's overhead, IDEA's interface is fast and responsive, even on a mid-level machine. It even behaves well with large projects with thousands of source files. The software should run on any platform with JDK 1.4 installed (a JDK 1.3–compatible version is planned), but the primary supported platforms are Windows, Mac OS, and Linux.

Figure 1 shows a typical shot of IDEA 3.0 in action. The interface should be comfortably familiar to anyone who has used an IDE before; nothing radical here. Apart from a toolbar and menu selections, IDEA 3.0 uses a series of tool windows that dock along the margins to provide access to such things as code structure, compilation messages, and debugging stack frames. These windows can

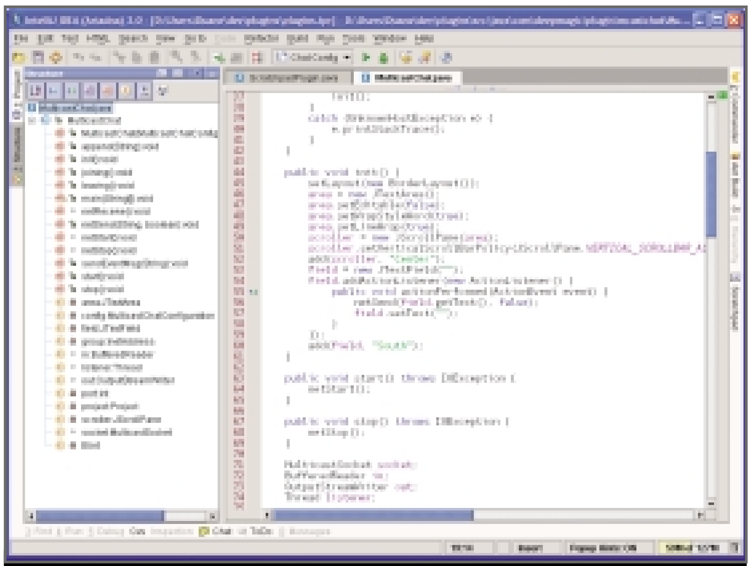


FIGURE 1 IDEA 3.0 in action

sys-con media

be moved around and displayed however you see fit. You can dock them, float them, make them slide in and out over your main window, and make them hide themselves when not needed.

As with most IDEs, you work on code as part of a project. A project includes your classpath, pointers to your source files, and build and run targets. Each project you define is stored as an XML file to keep on your local file system. One of IDEA's strong points is that it doesn't force you to set up your code structure any particular way, nor does it force you to use its internal build engine to compile. It's perfectly happy to call out to Ant to perform the build, for example. You can even choose to store source and library-path references relative to the project file, making it easy to share a project file with a team of developers or between machines.

Flexibility Is the Key to Happiness

IDEA 3.0 is as flexible as a three-legged rubber monkey. It's like no other IDE (or any other tool for that matter) I've encountered. You can customize everything about this program – from code formatting, syntax coloring, imports organization, and error highlighting to how your windows and tool bars are oriented. Perfect for particular programmers like myself!

Take the issue of spacing in your code. Unlike other IDEs that offer two or three spacing choices, IDEA 3.0 gives you literally dozens of options. As you can see in Figure 2, you can control every nuance of your code spacing style. Similar options are provided for controlling how braces, blank lines, and other stylistic choices are handled.

Your formatting options can be applied selectively to a section of code, an entire file, or even all the files in your project or directory. When you cut and paste code in the editor window, IDEA automatically formats the code appropriately, including inserting appropriate indentions and keeping everything nice and neat. IDEA's flexibility doesn't end at source-code formatting, of course. You can control the positioning of all the tool windows, the various aspects of code completion, coloring, and the entire collection of hot keys.

Refactoring Support

IDEA 3.0 would be an excellent IDE even if its flexible configuration and ease of use were its greatest assets. Its support for code refactoring is by far the most exciting feature. Refactoring, the process of continually improving your code and its structure, is one of those things that we all know is important, but we tend to slack off because it can be a pain to reorganize our code and class structure without breaking everything. Not so with IDEA. It supports many of the refactoring patterns discussed in Martin Fowler's seminal work on the subject, *Refactoring*. Some of the capabilities include:

- Changing a method's signature to include new arguments
- Renaming or moving classes, methods, and members
- Extracting selected items from a class into a new interface
- Introducing a variable from a selected expression
- Encapsulating field references into a method
- Pulling members up into a super class

IDEA makes it trivial to rename a class or move it to another package. Not only does it make the necessary changes to the code, it tracks down and corrects all the references to the class in Java code, import statements, JSP scriptlets, even Javadoc comments and XML files (like your Struts config file). In addition, it removes the old file from your version control system and adds the new one. It's so seamless that you don't think twice about moving a method from one class to another or reorganizing package structure as your project evolves. This is the first time we have such powerful tools in such a reasonably priced IDE.

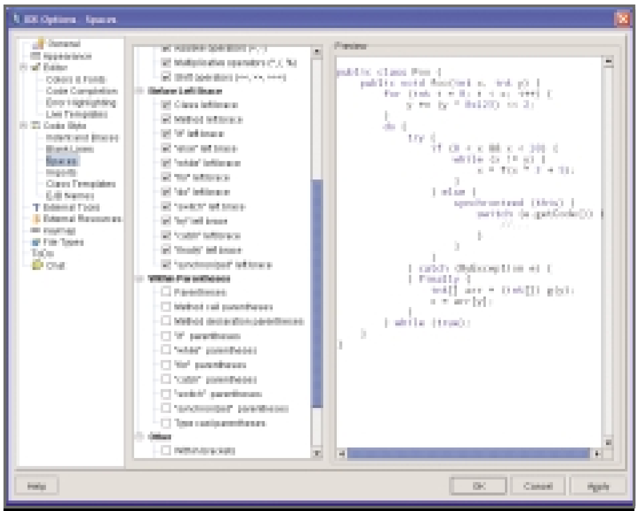


FIGURE 2 IDEA 3.0 source formatting options

J2EE Support

IDEA is way ahead of the game in terms of supporting JSP and EJB development. Most IDEs stop at syntax highlighting for JSPs, but IDEA adds much more. It allows you to define the roots of your Web applications in your source tree, providing internal awareness of your tag libraries, classpaths, and other properties. It can then perform code completion on JSP tags, bean properties, and even the file path include statements. It highlights Java errors in scriptlets and complains about invalid object references just as it does with Java source code. It generates EJB interfaces and allows full refactoring of them as well. JetBrains also promises integrated JSP and EJB debugging in the final release, but it was not yet available at the time of this writing.

Summary

I feel as if I haven't scratched the surface of everything this IDE can do and how well it does it. Overall, I'm very impressed by this as of yet unreleased product. Beginners will appreciate its good, easy-to-use editor, code completion, version-control system integration, and flexible configuration options, while advanced users will marvel at its seamless support for powerful refactoring operations and its rich plug-in API. Developers who are used to code wizards and GUI builders may be disappointed, however, as this tool does not attempt to address these areas.

Everything is wrapped up into a nice, reasonably priced package that most development shops should actually be able to afford. IntelliJ IDEA 3.0 will be available this fall.

Product Snapshot

- **Target Audience:** Java and JSP programmers
- **Level:** Beginner to advanced
- **Pros:** Flexible code formatting and interface, extensible API, first-class J2EE support, powerful code generation, Ant and JUnit support
- **Cons:** Limited support for non-Java source files, no GUI builder

Our World Live, Inc. to Demonstrate HIP 1.0 PJ and HIP 1.0 J2IP at JDJEdge West

(Huntington Beach, CA) – Our World Live, Inc. (OWL) will demonstrate its Human Interface Package (HIP) products for Personal Java (PJ), J2SE, and J2EE at JDJEdge 2002 West in San Jose, CA, October 2-3.

At the JDJ Partner Pavilion booth, OWL will show how its 100% Pure Java products, HIP v1.0 PJ and HIP v1.0 J2, transcend the limitations of Sun's AWT and Swing, enabling Java developers to develop advanced GUIs and reduce development time with out-of-the-box APIs for true freeform shapes, transparencies, and dragability. Show specials and giveaways will be offered.

"For the first time, Java applications can easily incorporate transparencies and freeform shapes, enabling more efficient use of limited screen real estate, providing a better user experience, and reducing development time," said Andreas Haas, OWL's CTO. Available online through OWL's Web site, HIP products can be licensed by developers, enterprises, OEMs, and educational institutions.

www.ourworldlive.com

Sun Unveils Java 3D API 1.3

(Santa Clara, CA) – Sun Microsystems, Inc., has brought new features and functionality to its Java 3D API graphics technology with the debut of Java 3D version 1.3. New features include configured universe and depth-sorted transparency capabilities, as well as advanced texture mapping and performance improvements.

www.sun.com

THOMAS GIVENTER WINS WEAKEST GEEK CONTEST

(New York) – Thomas Giventer (Giventer Software Systems, Ithaca, NY) was the grand prize winner of SYS-CON Media's Web Services Edge/JDJEdge 2002 East Weakest Geek competition, a Java, XML, and Web Services technology trivia contest. Giventer competed against fellow developers, the clock, and ***JDJ*** editor-in-chief Alan Williamson for the $10,000 prize, sponsored by Rational and IBM.

Six qualifying-round winners each received a copy of Rational XDE Professional and IBM WebSphere Studio Application Developer software.

www.sys-con.com

Parasoft Delivers Jtest to Complement Sun ONE Studio IDE

(Monrovia, CA) – Parasoft's Jtest is now available for integration with the Sun ONE Studio Integrated Development Environment. It automates key unit testing practices, such as white-box, black-box, and regression testing, and enforces more than 300 industry-respected coding standards through static analysis. It also has rules support for EJB components, Design by Contract (DbC), JSP technology, servlets, and project metrics.

www.parasoft.com

SITRAKA PARTNERS WITH BEANS FACTORY

(Toronto) – Sitraka has formed a new partnership with Beans Factory, a leading component technology enabler in Asia, to support Sitraka's growing Asia-Pacific client base. Beans Factory will initially be providing Sitraka JProbe training courses to customers in the Asia-Pacific region, offering onsite private training for development teams and public training sessions for individuals.

www.beansfactory.com
www.sitraka.com

BEA Challenges Developers to Build Reliable Web Services in Minutes

(San Jose, CA) – BEA Systems, Inc., has challenged developers to make the "WebLogic Workshop Web Services Connection" and see just how easy it is to build a Web service in minutes.

Contestants receive a free T-shirt and are entered into a drawing for one of three prizes, including a two-year subscription to **SYS-CON Media's *BEA WebLogic Developer's Journal***.

To participate, go to http://dev2dev.bea.com/products/wlwpromotion.jsp.

Manning Releases Java Development with Ant

(Greenwich, CT) – *Java Development with Ant*, by Erik Hatcher and Steve Loughran, shows both beginner and experienced users powerful and creative uses for Ant. The book emphasizes basic concepts, starting with Ant's XML-driven build process, and leads you step-by-step through everything you need to know to compile, test, package, and deploy an application. It then guides you through the maze of more complex situations common in larger projects such as enterprise Java applications and Web services.

www.manning.com

Sybase Introduces Borland JBuilder 7 Enterprise - Sybase Edition

(Dublin, CA) – Sybase, Inc., has announced the availability of the next-generation Java development environment, Borland JBuilder 7 Enterprise - Sybase Edition. The software simplifies EJB 2.0 development with two-way visual designers and rapid deployment to Sybase EAServer. It enhances developer productivity with UML code visualization, refactoring, unit testing, and documentation tools, and enables development and deployment of applications on Windows, Linux, Solaris, and Mac OS platforms.

Sybase has also announced EAServer 4.1, the first application server with J2EE 1.3 compatibility for AIX and HP-UX development platforms.

www.sybase.com

TogetherSoft Announces Plans to Acquire WebGain Technology

(Raleigh, NC / San Jose, CA) – TogetherSoft Corp. and WebGain, Inc., have announced that TogetherSoft plans to acquire WebGain Studio, which consists of VisualCafé, StructureBuilder, Business Designer, and Quality Analyzer and provides support for BEA WebLogic Server.

As part of the proposed agreement, TogetherSoft intends to provide full support for current WebGain Studio customers. TogetherSoft will also provide an integration package, offering WebGain Studio customers the opportunity to migrate to TogetherSoft's application development environment.

www.togethersoft.com

Exclusive: Excerpts from JavaDevelopersJournal.com

JDJ EDITOR INCENSED: ‘WHAT’S MICROSOFT UP TO?’ HE ASKS....

by JDJ News Desk

WILL MICROSOFT ever carry a truly current and compatible Java Virtual Machine in Windows XP? That’s the question Java developers worldwide began asking themselves the moment they heard of Microsoft’s announcement that it is reinstating the ability to run Java programs in Windows XP, as of the new release of the service pack this summer.

“I can only regard this as a ploy by the monopolist to avoid having Judge Whyte issue a preliminary injunction,” thunders JavaLobby founder Rick Ross, referring to the statement by Jim Cullinan, lead product manager for Windows, that, “For the next year and a half, Microsoft is going to include the JVM in Windows XP.”

Cullinan added: “Then we’ll make the changes to make sure that moving forward, we don’t put Windows or our customers at risk.” This prompted *Java Developer’s Journal* editor-in-chief Alan Williamson to comment: “It’s a great headline – ‘Java is back in Windows XP’ – but the small print stinks.”

Williamson continues: “It would appear that politicians aren’t the only ones with the ability to do a turnabout in the face of public opinion. Microsoft has announced that our dear friend Dukey is to return to Windows XP; however, the taste is bitter-sweet. They will be shipping their own JVM implementation, which if you remember, they were only licensed to use up to version 1.1.4.

“Sun has yet to comment on this,” Williamson notes. “But I am sure that, like me, Java developers everywhere will be incensed that the announcement has only gone so far, and not really far enough.

“Sadly this doesn’t take us any closer to the situation where we can ship our Windows users an executable JAR file with complete confidence that they have all the necessary software in place. Microsoft has made overtures that this Java addition will be only temporary, i.e., for the next 18 months only, and they will not be making any effort to update or fix any bugs, citing the Sun agreement as the reason why they can’t.

“You can’t help but be suspicious about the whole thing,” says Williamson. “Is Microsoft doing a sleight-of-hand trick? Are we to be watching the left hand for the time being, not aware of what the right hand is doing? What are they up to?”

JDJ’s J2ME editor, Jason R. Briggs, is also highly skeptical of Microsoft’s motives, observing that “Not only do they re-include their obsolete VM, thereby gaining some needed points with the courts (they hope), but they also manage to dilute the Java brand in the meantime.”

Briggs’ fear is that the general public will come to associate the term “Java” with software that may turn out to be insecure. “And since ostensibly they now ‘already have Java installed’,” Briggs explains, “the risk is that they won’t go and download Sun’s better/later model.”

Somewhat ruefully, Williamson concurs. “As with many Microsoft announcements,” he says, “more questions are usually asked than answered. This one is no different.”

Stay Focused

I think Sun has chosen a bad strategy to promote Java and J2EE ("Are You Ready to Rumble" by Michael Deasy [Vol. 7, issue 7]). They spend too much time pointing the finger at the "evil" Microsoft instead of focusing on the technology. Why is this a bad approach?

1. It looks as if there's nothing interesting about the technology if they put the main focus elsewhere (Microsoft, etc.). If in the introduction to the platform they quote the lawsuit against Microsoft, it makes me feel that the whole purpose of Java and J2EE is to kick Microsoft's ass – a "cheap" reason to live for such a cool platform, isn't it?
2. My experience is if I want to do perfect work, I have to stay focused on what I'm doing, not on finding out what's wrong with the others (except to avoid the same mistakes and make my work better). Which in this case means that instead of focusing on possibly relevant but definitely unconstructive criticism, they should make sure the crucial presentation is perfectly prepared (referring to the failed connectivity example in Question #3). Everybody should agree the second is harder to accomplish, and I truly hope that was not the main reason to focus on the first.

Igor Koziak
ikoziak@montage.ca

Sun Is Java's Worst Enemy

J2EE blows .NET away completely on technical merits alone. The depth, richness, and reliability of this platform are awesome in comparison to .NET, which is a version 1.0 platform with an aggressive marketing strategy. Why would Sun choose to promote Java based on corporate cultural differences? Nobody cares! Java should have been represented by IBM. At least they understand what J2EE is and why it is by far the best solution.

Ted Barbusinski
tbarbusinski@stellcom.com

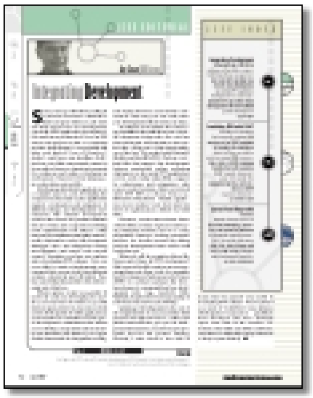
Good Reference

"Programming Restrictions in EJB Development" by Leander van Rooijen (Vol. 7, issue 7) is an excellent article. It provides a good reference for EJB developers. I was confused about EJBs and the helper classes related with thread usage; this article made it a lot clearer.

Veerendra Shukla
sveerendra@netpace.com

Stop Comparing Apples to Oranges

I get a little tired of people who equate vi with a development environment ("Integrating Development" by Ajit Sagar [Vol. 7, issue 6]). When was the last time you used vi on Windows? The fact is, vi is used in the "Unix development environment." Unix was and still is an OS written by programmers, for programmers. It constitutes a development environment with a powerful set of tools (grep, X-Windows, shell scripting, make, etc.).



Please, let's stop comparing apples to oranges, and compare the various IDEs to the Unix environment.

Ron Theriault
ron-t@austin.rr.com

Doing Business with the Opposition

I was just wondering why Steve Ballmer is interested in doing business with the opposition ("Java in a Flash!" by Alan Williamson [Vol. 7, issue 7])? I don't know how wise it was to go to Redmond. With .NET, Microsoft is trying to survive in the server networking industry. Sun in their wisdom has known all along that the network is the computer!

Major question: Will Microsoft make distributed computing a more friendly place for all?

James Ruvoo
jruvya@yahoo.com

Under the Hood

"Hello World! in 70 Bytes" by Norman Richards (Vol. 7, issue 7) is perfect for those readers who really want to understand what happens under the hood of the JVM.

Mike Morris
mmorris@topcoder.com

JDJEDGE CONFERENCE & EXPO FINAL PROGRAM INSIDE
JAVA DEVELOPER'S JOURNAL
WEB SERVICES WORLD TOUR DON'T MISS IT!
JAVA DEVELOPER'S JOURNAL
The World's Leading Java Resource
ZX
Spectrum Emulator in Java
COMING TO A CITY NEAR YOU

Training Days

How (and where) to brush up on your skills

WRITTEN BY
BILL BALOGLU &
BILLY PALMIERI

It's no secret to anyone who works in the technology industry that continous training (and retraining) is required. The only thing that's constant in this business is change, and engineers need to be ahead of the curve on the latest and greatest technologies.

The engineers we work with understand that ongoing training is a fact of life. There are many different ways to get this training, and some of our associates recently shed some light on the benefits and drawbacks of each type.

Frank is a veteran and proponent of vendor-based training seminars, having taken courses offered by such companies as Sun and Microsoft. These on-site courses require time off from his usual assignments, so he tries to fit them in between contracts.

"The biggest advantage to these kinds of courses is that you're going straight to the source," he says. "If the people who built the application aren't the experts, who is? On the other hand, they're also evangelists for the product, so you won't get a critical perspective that you might get from a neutral instructor."

Frank also likes interacting with the other people at the training sessions, as they bring a variety of experiences and perspectives. "I've made some good networking contacts and friends taking these courses," he says. "And most people seem to be at similar skill and experience levels."

There tends to be a lot of lab time in these courses and the chance for interaction with the instructor and other participants can make for an in-depth learning experience. "You just don't get that from a book or a CD-ROM," says Frank.

Mike, a senior engineer, has taken plenty of vendor-sponsored courses, but he's also had good experiences with training that's approved (but not taught) by the vendors. Companies like New Horizons offer classes in major brand-name technologies, "and the instructors can be objective about the products," he says.

"It helps to know the problems with an application or a technology as you're learning how to use it," says Mike. "The instructors can talk about the relative pros and cons of different companies' technologies in this atmosphere."

Most of these courses offer intensive three-to-five day seminars. "You need to stay focused and sharp and take good notes," says Mike. "The intensive schedule is good for me since I'd rather be working on a project than taking days off. But you don't get as much hands-on lab time as you'd get in a longer course.

"If you can get an employer to cover the cost of this kind of training, go for it," says Mike. "The schools know how much training can add to an engineer's hourly rate, so some of the courses can be very expensive."

Julie has a different view of training based on her own learning style. "I just don't learn as effectively in a crash course as I do in a long-term one," she says. Which is why she prefers taking classes at local colleges.

"A night class that meets two or three nights a week works well for me," she says. "I can take them while I'm on a project and it makes a nice break from my daily routine. A long-term course could be a problem for someone who travels a lot, but I don't."

State and community college courses are much more economical than corporate training, and Julie also enjoys interacting with the other students. "Almost everyone in the class has a day job, so we're all in the same boat," she says. "It's a good chance to network.

"I also like the chance to work long term with the instructors," she says. In the past year she's seen an improvement in the quality of instruction at the local college level. "In the past, most instructors had academic backgrounds. But a lot of people have left private industry to teach, so now they have more real-world industry experience."

Tim is an engineer who thrives on multiple contracts and a lot of traveling, so most traditional courses are not an option. "I'm always on the road, so when I need to pick up a class I do it online," he says.

"A lot of them now have live streaming interaction with audio and online chat with the instructors," he says. "For interactive classes I do need to be at my laptop at a certain time, but it's kind of cool to be taking a class while sitting in a hotel room or waiting in an airport."

Tim admits that modem speed and bandwidth issues can be a problem when accessing an interactive course on the road, but in his case, the flexibility outweighs the challenges. "The content of some of these classes is archived so I can go back and review something I might've missed or forgotten the first time around," he says.

"Online training isn't a perfect system yet," says Tim. "But I like how I can complete courses without taking time away from my travel or work."

There are clearly advantages and disadvantages to each type of training, but the most important thing is to choose a course that fits your lifestyle, schedule, and learning style.

Whether you value "from the horse's mouth" vendor training, the objective neutrality of a corporate school, the old-school comfort of a college setting, or the flexibility of an online course, the most important thing is to keep your skills up-to-date and on the cutting edge.

jdjcolumn@objectfocus.com

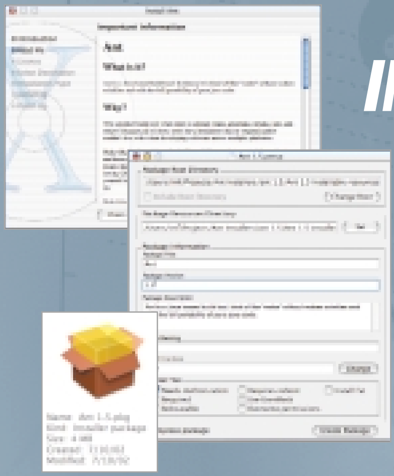
AUTHOR BIOS
Bill Baloglu is a principal at ObjectFocus (www.ObjectFocus.com), a Java staffing firm in Silicon Valley. Bill has extensive OO experience and has held software development and senior technical management positions at several Silicon Valley firms.

Billy Palmieri is a seasoned staffing industry executive and a principal at ObjectFocus. His prior position was at Renaissance Worldwide, where he held several senior management positions in the firm's Silicon Valley operations.

JAVA DEVELOPER'S JOURNAL ADVERTISERINDEX

ADVERTISER	URL	PHONE	PAGE
AccelTree	www.acceltree.com/order.asp		33
Actuate Corporation	www.actuate.com/info/jbjad.asp	800-884-8665	51
Altova	www.altova.com		31
AltoWeb	www.altoweb.com		21
Apple Computer, Inc	www.apple.com/xserve		4-5
BEA	dev2dev.bea.com/useworkshop		6
Borland Software Corp.	www.borland.com/new/jb7/5068.html	800-252-5547	23
Capella University	www.capella.edu/dsk	888-CAPELLA ext. 6024	39
Crystal Decisions	www.crystaldecisions.com/cr9/002/	888-333-6007	13
Engenuity Technologies	www.jloox.com	800-684-5669	45
ESRI	www.esri.com/arcims	888-289-5084	35
Fiorano Software	www.fiorano.com	800-663-3621	59
HiT Software	www.hitsw.com/xmlrdb	408-345-4001	55
IBM	ibm.com/developerWorks/linux/cd		28-29
IBM	ibm.com/db2/rocks		41
IIT Software GmbH	www.swiftmq.com		50
Improv Technologies	www.improv-tech.com/jdj/download		65
InetSoft Technology Corp.	www.inetsoft.com/jdj	888-216-2353	63
Infragistics, Inc.	www.infragistics.com	800-231-8588	14-15
INT, Inc	www.int.com	713-975-7434	12
iSavvix	http://community.isavvix.com		69
Ittoolbox	www.Ittoolbox.com		71
Java Developer's Journal	www.sys-con.com/java		75
JBOSS	www.jboss.org		58
JDJ Store	www.jdjstore.com		79
Jinfonet	www.jinfonet.com/jdj9.htm	301-838-5560	47
Macromedia	www.macromedia.com/go/jrun4jdj		37
Motorola	www.motorola.com/developers/wireless		8
\n software inc.	www.nsoftware.com		49
n-ary	www.javaSOS.com		73
Northwoods Software	www.nwoods.com/go/	800-434-9820	70
Oracle Corp.	www.oracle.com/ad	800-633-1072	17
Our World Live			105
Parasoft Corporation	www.parasoft.com/jdj9	888-305-0041	43
Precise Software	www.precise.com/jdj	800-310-4777	27
QUALCOMM Incorporated	http://brew.qualcomm.com/ZJD4		53
Rational Software	www.rational.com/offer/javacd2		11
Sitraka	www.sitraka.com/jclass/jdj	800-663-4723	19
Sitraka	www.sitraka.com/jprobe/jdj	800-663-4723	57
Sitraka	www.sitraka.com/performasure/jdj	800-663-4723	106
Sonic Software	www.sonicsoftware.com/jdj	800-989-3773	2
Spiritsoft	www.spiritsoft.com/climber		25
Zero G	www.zerog.com	415-512-7771	3, 67

What's in the next issue of *JDJ*?



BUILDING INSTALLERS FOR OS X

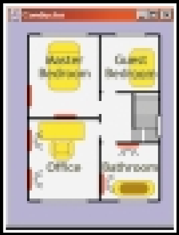
Java development on OS X is very similar to Java development on any platform, particularly any Unix platform. This article shows you how to package your Java application into a native OS X installer using one of the many free development tools that come with the platform.

TIERING INTO J2EE APPLICATIONS

What are the challenges in developing a J2EE-based application? This article discusses EJB design practices and Web component development.

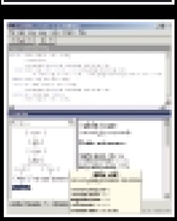
MANAGING JAVA SOURCE CODE DEPENDENCIES FOR SCM

There are many facets to consider in the implementation of even the most basic Software Configuration Management (SCM). For Java, with its import mechanism, these simple goals often become unmanageable when the source code tree grows beyond a certain point of complexity. This article discusses the underlying relationships that make even basic Java SCM problematic and how to manage them.



PLUG IN YOUR COMMAND PROCESSOR NOW AND START SAVING MONEY!

The Command Processor tool takes a Java object and creates a command-line interface to its public methods. These public methods are essentially your Application Programming Interface (API). During the course of this article we'll get a good look at the java.lang.reflect package, as well as kick the tires on the Regular Expression package included in the 1.4 JDK.

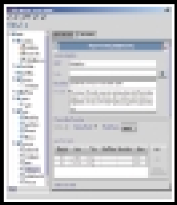


WHOLE HOUSE AUDIO IN THE PALM OF YOUR HAND *PART 3*

How a Java application can easily take control of physical systems with the right hardware, both controlling hardware and responding to real-world events (well, a doorbell).

ASK DR. JAVA

Prescriptions for your Java ailments. Answers to your Java questions.



iostream of Consciousness

WRITTEN BY
BLAIR WYMAN

I've been weaving these threads of cubist pseudo-consciousness for over a year now, and the consequences of such promiscuous international celebrity are really starting to get out of hand.

It seems as if I'm constantly handling "product placement" e-mails from big-shot marketing types, begging me to scrupulously avoid *any* mention of their products *whatsoever*. (They always add that "whatsoever" in there, as if I'd try to sneak something in on them. Well, you don't have to hit *me* over the head with an Uncle Bernie's Fine Cavalla mackerel!)

I've been up to my scalp in Java code again this month and have enjoyed many opportunities to reflect on its unique strengths and weaknesses. Most of these wool-gathering sessions end rather abruptly, as a build completes, a button needs pushing, or another blankety-blank test case blows bitwise chunks all over the logs. Reality intrudes with an indelicately echoed Ctrl-G, and it's another trip to the top of the E/C/D waterfall (and me without my barrel). I can't believe they pay me to have all this fun.

One thought that keeps recurring during these flights of javac-induced fancy is the very different nature of the bugs I find myself chasing and squishing in Java, compared with the other languages I've used (and I've used a few).

At the risk of dropping my pretentiously thin veil of false humility – the one you've seen through since day one – I must say I've cranked out some serious code since becoming a professional programmer, and there's no end in sight. Some of the popular languages I've used (and abused) include REXX, MASM, Tcl, awk, gawk, Perl, Pascal, Modula-2, C, C++, csh, and my favorite, Java (of course). At best, I've "mastered" maybe one or two of these languages at some point over the years. (Sometimes I imagine that Java belongs on my "mastery" list…then I wake up.)

Way back before this whole overblown "microprocessor" fad perturbed the orderly Hollerithic landscape of fanfold batch computing, I even programmed something called an "electronic analog" computer (and badly, at that).

Electronic analog computers are "programmed" using patch cords and panels, and I was not without innate skills in that area, even in high school. In fact, I was a natural. I'm here to tell you I could plug and unplug those patch cords lickety-split, thank-you-very-much.

Unfortunately, since I didn't have any idea which plugs to patch where, my results *stunk* (often literally, wafting an aroma of melting plastic insulation). I guess I should have paid attention to that Calc II prerequisite, after all.

With apologies to Henry Spencer (author of the "Ten Commandments for C Programmers"), perhaps there is an analog computing axiom/corollary/lemma for would-be OO programmers: "Be careful how you hook your objects together, lest they stink." Yeah, that quote will ensure me a spot in history. You betcha.

Anyway, analog computers are really quite remarkable. If you patch together the right combination of plugs, you can accurately and continuously model complicated parallel mathematical equations of many variables. To the extent that these mathematical equations correctly model an "analogous" physical system, the electronic system behaves just like that leaf spring, or that ambient vapor pressure, or that trout population, or that ICBM nose cone. (If you're interested in finding out more about analog computing, try dropping the phrase "analog computer" into your favorite search engine or visit Doug Coward's wonderful "Analog Computer Museum" Web site, http://dcoward.best.vwh.net/analog/.)

So far, I've yet to chase a pointer bug in Java, but that's not to say that it's all a bed of roses. For my part, I've found Java's principle sanity thief usually extends java.lang.Thread or implements java.lang.Runnable; thread synchronization bugs can be deucedly difficult to dispatch or dismember.

I remember a piece of multithreaded Java code missing some synchronization in a chained assignment statement:

```
inNdx = outNdx = 0;
```

It's around 4:00 p.m. on Friday when the call comes in: "We've encountered an ArrayIndexOutOfBoundsException that stinks like burning insulation. You didn't let Wyman near this code, did you?"

blair@blairwyman.com

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

Rick Ross deposition
Mr Ross, how long would it take to download Java?
I would need more detail to give an accurate answer
RICK ROSS
You're avoiding the question Mr Ross, did you or did you not download the JRE in question?
Sorry I thought you were asking...
RICK ROSS
www.vladkolarov.com
Don't interrupt Ross. Your honour, this is a hostile witness and I'd like declare this case against Starbucks a mis-trial.
This guy is good!
BALLMER
Bill

our world live

ourworldlive.com

sitraka

www.sitraka.com