SYS-CON

JAVA CERTIFICATION PROS + CONS PGS. 40 & 62

No. 1 *i*-Technology Magazine in the World

JDJ

OCTOBER 2004 VOLUME:9 ISSUE:10

Integrating with Eclipse

Page 24

JAVA GAMING

Understanding the basic concepts

PART 1

PLUS...

- An Introduction to Service Data Objects
- Best Practices for JDBC Programming
- Java Object Serialization for Performance
- Managing Objects Between Java and C

RETAILERS PLEASE DISPLAY
UNTIL DECEMBER 31, 2004
$9.99US $9.99CAN
11>
0 74470 01751 6

Oracle Application Server

"Oracle Portal is a powerhouse"

***Network Computing*, 5/13/04**

	Oracle AS 10*g* Portal	IBM WebSphere Portal 5.0	Microsoft Office Share Point Portal Server 2003	BEA WebLogic Portal 8.1
Customization (35%)				
Development framework (20%)	5	3	3	3
User personalization (15%)	4	4	4	2
Integration (30%)				
Collaboration/productivity (10%)	4	4	4	4
Prebuilt integration (10%)	4	4	4	4
Identity management (5%)	4	4	4	4
Search/document management (5%)	4	4	4	4
Architecture (20%)				
Framework (5%)	4	4	4	4
Implementation (5%)	4	4	4	4
Manageability (5%)	4	4	4	4
Standards (5%)	4	4	4	4
Security Features (10%)	4	3	3	2
Price (5%)	5	3	5	4
TOTAL SCORE (100%) Total scores and weighted scores are based on a scale of 0-5.	4.25	3.45	3.40	2.70

Oracle Application Server Portal 10*g*
Winner of the 2004 Network Computing Editor's Choice Award

ORACLE®

Network Computing Editor's Choice

oracle.com/portal
or call 1.800.633.0759

Source: Network Computing, 5/13/04, Well-Connected Awards, p. 92

JAVA DEVELOPER'S JOURNAL

Editorial Board
Desktop Java Editor: Joe Winchester
Core and Internals Editor: Calvin Austin
Contributing Editor: Ajit Sagar
Contributing Editor: Yakov Fain
Contributing Editor: Bill Roth
Contributing Editor: Bill Dudney
Contributing Editor: Michael Yuan
Founding Editor: Sean Rhody

Production
Production Consultant: Jim Morgan
Associate Art Director: Tami Beatty-Lima
Executive Editor: Nancy Valentine
Associate Editors: Jamie Matusow, Gail Schultz
Assistant Editor: Natalie Charters
Online Editor: Martin Wezdecki
Research Editor: Bahadir Karuv, PhD

Writers in This Issue
Derek Ashmore, Calvin Austin, John Cahill, Brent Daniel, BIll Dudney, Mike Edwards, Tim Ellison, Naveen Gabrani, Jeremy Geelan, Hari K. Gottipati, Chet Haase, Onno Kluyt, William Knight, Ajit Sagar, Dmitri Trembovetski, Kevin Williams, Joe Winchester, John Zukowski

To submit a proposal for an article, go to http://grids.sys-con.com/proposal

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

888 303-5282
201 802-3012
subscribe@sys-con.com

Cover Price: $5.99/issue. Domestic: $69.99/yr. (12 Issues) Canada/Mexico: $99.99/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: 201 782-9638

Java Developer's Journal (ISSN#1087-6944) is published monthly (12 times a year) for $69.99 by SYS-CON Publications, Inc., 135 Chestnut Ridge Road, Montvale, NJ 07645. Periodicals postage rates are paid at Montvale, NJ 07645 and additional mailing offices. Postmaster: Send address changes to: Java Developer's Journal, SYS-CON Publications, Inc., 135 Chestnut Ridge Road, Montvale, NJ 07645.

Worldwide Newsstand Distribution
Curtis Circulation Company, New Milford, NJ
For List Rental Information:
Kevin Collopy: 845 731-2684, kevin.collopy@edithroman.com
Frank Cipolla: 845 731-3832, frank.cipolla@epostdirect.com

Newsstand Distribution Consultant
Brian J. Gregory/Gregory Associates/W.R.D.S.
732 607-9941, BJGAssociates@cs.com

Is Mergermania Back?

Jeremy Geelan

Hurricanes Ivan, Charley, and Frances notwithstanding, sometimes being in the eye of the storm has its advantages.

At SYS-CON Media, where we by definition dwell at the epicenter of what might be called the *i*-technology weather cycle, our central position allows us to ask industry influencers for quickfire responses to burning issues of the day.

The recent PeopleSoft–Oracle decision handed down by the U.S. Justice Department is a case in point. U.S. District Judge Vaughn Walker had given his verdict on the antitrust lawsuit filed seven months ago by the U.S. Justice Department and 10 states seeking to block Oracle's would-be "hostile takeover" of rival PeopleSoft. In his 164-page decision Walker ruled that there weren't sufficient grounds to block it.

The judgement was enough to send PeopleSoft's stock climbing. While it seemed to bring to an end a lawsuit on which Oracle has allegedly spent some $60 million and PeopleSoft even more, some $70 million, we wondered whether it also marked a new beginning: the reemergence of mergermania in the world of enterprise technology. Also, if Oracle "won," who lost?

Taking soundings around us, it wasn't long before we heard back from Mitchell Kertzman, now of Hummer Winblad Venture Partners – and famous before that, of course, as the founder and CEO of Powersoft in 1974, which merged in February 1995 with Sybase, of which he subsequently became chairman of the board and CEO.

"The U.S. Justice Department's patchwork-quilt antitrust policy is the loser here," Kertzman commented. "The software industry needs a clearer, more coherent statement and practice of antitrust guidelines from Justice," he added.

Another prime mover in the enterprise technology space, Dr. Adam Kolawa – cofounder, chairman, and CEO of Parasoft – felt that the problem lay in the issue having ever gone to court in the first place. "I think the suit was a mistake," Dr. Kolawa declared. "Oracle should have been allowed to purchase PeopleSoft in the first place. So we wasted some time and money and now we are able to get down to business."

And the "winner"? Not Oracle, according to the Parasoft chairman. "In a situation like this," he contended, "the winner will be the other guy, not Oracle. Oracle will have to support PeopleSoft customers for quite a while. It cannot just kill the products. In the meantime, because of all the publicity, anybody who thought about buying PeopleSoft software will already have decided to buy competitive packages. The real winners are probably SAP and IBM."

Dr. Kolawa offered a final thought: "I have seen battles like this before, and they were irrelevant a few years later. Do you remember Compaq buying DEC, or HP buying Compaq? They brought more headaches than they were worth to the companies that were involved in them."

The Powersoft merger with Sybase in 1995 was, at the time, the most valuable in the history of the software industry. But an Oracle–PeopleSoft deal would set a new record, including a record for the hostile nature of the whole arrangement.

In a letter written on Sept. 9 after the U.S. District Judge's decision, Oracle chairman Jeff Henley and CEO Larry Ellison wrote to the PeopleSoft board: "With the removal of the U.S. antitrust issue and Oracle's commitment to acquire PeopleSoft, we are hopeful that a transaction can occur." PeopleSoft's reply was curt: "PeopleSoft's Board has carefully considered and unanimously rejected each of Oracle's offers, including its current offer of $21 per share. On May 25, 2004, the Board concluded that the current offer was inadequate and did not reflect PeopleSoft's real value."

To our mergermania question, JBoss CEO Marc Fleury commented: "Consolidation is a natural fact in the software markets. I don't see losers but rather efficiencies at play." He went on to note that the mergers and consolidations might well be catalyzed by the recent rise of open source software. "Open source accelerates this natural consolidation by putting great pressure on the profits of infrastructure software players, for example, the rumors have it that BEA is next on Oracle's shopping list," Fleury said.

No one said 2004 would to be a dull year in the enterprise technology world.

Jeremy Geelan is group publisher of SYS-CON Media, and is responsible for the development of new titles and technology portals for the firm. He regularly represents SYS-CON at conferences and trade shows, speaking to technology audiences both in North America and overseas.

jeremy@sys-con.com

Solving the Toughest XML Data Integration Challenges!

Free 30-day Trial Download Now!

Simplify your XML Development with Stylus Studio XML IDE

Introducing Stylus Studio 6 XML Professional Edition

Tired of the complexities of advanced XML data integration? New and improved **Stylus Studio 6** washes away even the most stubborn data integration problems with a bold new formula that includes such active ingredients as **Convert to XML™**, a powerful utility for creating XML documents from any flat file data source. Support for **XSLT 2.0** and **XQuery 1.0** in Stylus Studio's leading XML editors, debuggers, and performance profilers. An all-new **XML Schema Editor** for visually developing XML data models. An **XML Grid View** that lets you work with tabular data as XML. Usability enhancements to our **XML Mapper**, and much more.

STYLUS STUDIO

Stylus Studio XML IDE.
Revolutionizing the way XML data integration gets done.

XML Productivity Through Innovation.

www.stylusstudio.com

OCTOBER 2004 VOLUME:9 ISSUE:10

JDJ contents

JDJ Cover Story

Feature

Java Object Serialization for Performance
by John Cahill

JDJ (ISSN#1087-6944) is published monthly (12 times a year) for $69.99 by SYS-CON Publications, Inc., 135 Chestnut Ridge Road, Montvale, NJ 07645. Periodicals postage rates are paid at Montvale, NJ 07645 and additional mailing offices. Postmaster: Send address changes to: JDJ, SYS-CON Publications, Inc., 135 Chestnut Ridge Road, Montvale, NJ 07645.

John Zukowski

Where's Java Going with 6.0?

With the release of the newly renamed Java 5.0 J2SE platform, it's time to speculate on just what might be coming in Java 6.0. Given the typical 18–24 month cycle for major J2SE releases, you need to think beyond the norm and not just about new specification releases that require updated versions in the platform.

Sun's Bug Database (http://bugs.sun.com/bugdatabase/) is a good source for ideas about the more normal Request for Enhancements (RFEs), what's missing from the current release or has been missing in past releases. Since Sun has a top 25 list, we won't repeat what's there. Also, obvious additions/updates such as for "new" technologies that Sun is promoting, e.g., the Sun Java Desktop System, will be left off.

Instead, let's consider how changes in the world and the Internet will lead to even more technologies being incorporated into the standard Java platform.

Hopefully, by the time Java 6.0 comes out, the patent on GIF writing will have timed out everywhere in the world and Sun can add this to the platform. Of course, why anyone would want to write GIFs any more can always be asked.

It's also about time Sun offered full support for PDF files. This is no small feat, as the printing API includes a PDF flavor, but no native printing support. This added library would need to offer support for parsing, generating, searching, previewing, printing, encrypting, and annotating. Form support would also be beneficial, though that could wait for 7.0.

Instant messaging and peer-to-peer networking are very popular these days. Expect to see APIs like Jabber (www.jabber.org) for XML streaming protocols and support of the up-and-coming Extensible Messaging and Presence Protocol (XMPP) from the Internet Engineering Task Force (IETF) available.

A new javax.blog package could serve as a hotbed for serving up personal diaries. With the help of the XMPP capabilities, support for having everyone's blogs automatically intertwined by just starting a standard blog server could be available. Group blogs would need to be supported, as well as a handful of Web services that, for example, enable Amazon associates to maximize revenue options.

Because of Josh Bloch's huge contribution to the Java platform and to maximize his new stock options over at Google, Sun should include the Google Web API service (www.google.com/apis/) in the standard SDK. With broadband everywhere by then, developers could easily add features like Google's spell-checker, translation options, or just put AdSense in every available Web application.

Viruses and spyware are rather popular these days. With Sun's renewed interest in the desktop, look for new API hooks into the anti-virus software on the client. Expect the option to automatically check for viruses on standard socket connections. To help virus writers, Sun could add a new virus writer/spyware toolkit to the platform. This would include features for polymorphic virus generation and keyboard capture. Also, look for an API that provides the ability to add users to and remove users from the operating system and to "su" over to another user to execute a command under the guise of that user.

With Sun and Microsoft settling their legal differences, and Microsoft no longer updating Internet Explorer's pseudo virtual machine for Java, Sun could include a pluggable Internet Explorer as a full-fledged browser in the JEditorPane/HTMLEditorKit, reversing the plugin debate.

For the low-carb developers out there, Sun should delete every deprecated field and method. This would lighten up the APIs considerably in some areas and remove many methods that just redirect calls to different methods. Of course, this would mean that methods like System.getenv(String) couldn't have been undeprecated in 5.0. But then, getEnv() might have been a better name had Sun not had to keep to the original name when the method got undeprecated.

For the high-carb developers there, Sun should finally add operator overloading. Just look at how Groovy does this at http://groovy.codehaus.org/operators.html. Define a method like plus() in your object and you can do a + b. Define a next() method and use auto increment operators (++). What's wrong with mapping method calls to operators? You should also be able to define new operators like ^^ for power (since ^ is bitwise-or), like 2^^3 for 8.

With all these things missing from Java for so long, you wonder how it has even made it to version 5.0, let alone to the second release, 1.1. We'll just have to wait and see what Sun decides to add.

John Zukowski is working with SavaJe Technologies on a Java platform for mobile phones and is wrapping up his latest book *Definitive Guide to Swing for Java 5*.

jaz@zukowski.net

SYS-CON MEDIA

President and CEO:
Fuat Kircaali *fuat@sys-con.com*
Vice President, Business Development:
Grisha Davida *grisha@sys-con.com*
Group Publisher:
Jeremy Geelan *jeremy@sys-con.com*

Advertising
Senior Vice President, Sales and Marketing:
Carmen Gonzalez *carmen@sys-con.com*
Vice President, Sales and Marketing:
Miles Silverman *miles@sys-con.com*
Advertising Sales Director:
Robyn Forma *robyn@sys-con.com*
Sales and Marketing Manager:
Megan Mussa *megan@sys-con.com*
Associate Sales Managers:
Kristin Kuhnle *kristin@sys-con.com*
Dorothy Gil *dorothy@sys-con.com*

Editorial
Executive Editor:
Nancy Valentine *nancy@sys-con.com*
Associate Editors:
Jamie Matusow *jamie@sys-con.com*
Gail Schultz *gail@sys-con.com*
Assistant Editor:
Natalie Charters *natalie@sys-con.com*
Online Editor:
Martin Wezdecki *martin@sys-con.com*

Production
Production Consultant:
Jim Morgan *jim@sys-con.com*
Lead Designer:
Tami Beatty-Lima *tami@sys-con.com*
Art Director:
Alex Botero *alex@sys-con.com*
Associate Art Directors:
Louis F. Cuffari *louis@sys-con.com*
Richard Silverberg *richards@sys-con.com*
Assistant Art Director:
Andrea Boden *andrea@sys-con.com*

Web Services
Vice President, Information Systems:
Robert Diamond *robert@sys-con.com*
Web Designers:
Stephen Kilmurray *stephen@sys-con.com*
Matthew Pollotta *matthew@sys-con.com*

Accounting
Financial Analyst:
Joan LaRose *joan@sys-con.com*
Accounts Payable:
Betty White *betty@sys-con.com*
Account Receivable:
Shannon Rymza *shannon@sys-con.com*

SYS-CON Events
President, SYS-CON Events:
Grisha Davida *grisha@sys-con.com*
National Sales Manager:
Jim Hanchrow *jimh@sys-con.com*

Customer Relations
Circulation Service Coordinators:
Edna Earle Russell *edna@sys-con.com*
Linda Lipton *linda@sys-con.com*
JDJ Store Manager:
Brunilda Staropoli *bruni@sys-con.com*

Ajit Sagar
Contributing Editor

Take Two Patterns and Call Me in the Morning

Life is not easy for today's enterprise application architects. In today's IT world, the architect not only has to design solutions for a plethora of interdependent systems (as is obvious from the job description and title), he or she also has to conform to the ever-evolving standards in a shorter API life cycle, plan for the not-too-distant future, collaborate with business and technical environments, and work on a feasible roadmap for his or her application/application portfolio.

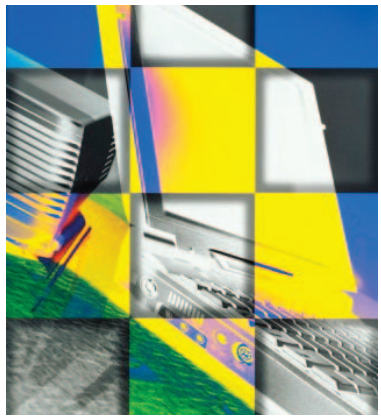
In large organizations, standards for various facets of business and technology are already laid out and managed between several competency centers/centers of excellence, and strict governance is often in place to ensure consistency throughout the organization. To successfully deliver products against hard deadlines, the architects have to make sure that everything complies in the governance process.

With the advances in software engineering, especially in component-based and service-oriented architectures, common guidelines have emerged in the form of design patterns. These design patterns help architects leverage what others have learned in their software design journey. The Java platform is an obvious example of the application of design patterns. Before distributed platforms such as Java came along, the number of folks who could spell "design pattern" was limited to the few who had read the Gang-of-Four book, and that was just a handful of developers in any organization. With Java – Listeners, Proxies, Observers, Factories, Delegates, Facades – all these became a part of the designers' common vocabulary. Reference architectures, frameworks, and plug-and-play followed soon after. Although Java is not solely responsible for this, it has definitely played a big part in the promotion of these concepts. Add UML and RUP to the mix and you not only have the toolkit, but also the ability to document and manage your application's development in a common way.

With all these wonderful tools in the architect's toolkit, why is the job still so complex? Given the tools at hand, an application architect should easily be able to develop applications that leverage:

- Application frameworks
- Architecture blueprints
- Adaptive architectures
- Reference architectures
- Prescriptive architectures

These architectures and frameworks have been developed in the software community, as well as within organizations, and they facilitate the design of applications from a common base and building blocks. Apply them to your application and presto – you have a two-minute recipe to create an instant product. Where's the catch?



The basic problem is that the guidelines and patterns, though invaluable, are created outside the application domain. Although they do address the needs of applications, that need is addressed across a number of applications. After all, it's the only way that reuse can be promoted. In this editorial we are focusing on the application architect. The architects in competency centers and standards groups focus on promoting reuse. The focus of the application architect is on leveraging reusable components and leveraging documented patterns to solve a business problem. However, unless a clear path is laid out for navigating through the available choices, he or she may easily choose to develop alternatives in order to meet the demands of the application.

This impasse between the prescription of reuse and its feasibility in the application's context needs to be carefully addressed. Just as the design principles that apply to a broad range of applications are made available for the applications, a guidebook that navigates through these principles should be developed for the application/application portfolio. This guidebook should focus on application design, treating the application as the center of the universe. Then clear governance practices should be established around exporting the reusable artifacts, learnings, and application patterns that emerge out of each application design cycle. These should then be incorporated into the rationale for selecting design patterns published by groups external to the application. After all, communication is a two-way street. Design patterns are no exception.

Ajit Sagar is a senior technical architect with Infosys Technologies, Ltd., a global consulting and IT services company. He has been working with Java since 1997, and has more than 15 years' experience in the IT industry. During this tenure, he has been a programmer, lead architect, director of engineering, and product manager for companies from 15 to 25,000 people in size. Ajit has served as *JDJ*'s J2EE editor, was the founding editor of *XML-Journal*, and has been a frequent speaker at SYS-CON's Web Services Edge series of conferences. He has published more than 75 articles.

ajitsagar@sys-con.com

"The basic problem is that the guidelines and patterns, though invaluable, are created outside the application domain"

The right Java, whatever the gig.

Borland® JBuilder.® The #1 Java™ tool in the world for a reason. Pick the size that fits your needs. Automate the routine stuff. Handcraft the unique. Have an active voice at every stage in the process. Move faster, and make every project a hit. Whether your application is headed to the Web, enterprise, or mobile, just pick the feature set you need. And start rockin'!

Customizable code editor • Refactoring • Local and remote debugging • Integrated unit testing • Two-way visual Struts designer • JSP™ tag library/ framework support • XML and Web Services • Mobile application development • Advanced build and configuration management with Apache® Ant • Visual EJB™ designer • Two-way deployment descriptor editor • Archive builder • Integration with all major J2EE™ application servers

go.borland.com/j6

Borland®
Excellence Endures™

Learn from the experts at the premier event for technical education – the 2004 Borland Conference – http://connect.borland.com/borcon04s

An Introduction to Service Data Objects

Integrating relational data into Web applications

by Kevin Williams and Brent Daniel

Late last year, IBM Corp., and BEA Systems, Inc., introduced Service Data Objects (SDO), a new data programming specification that complements existing Java 2 Enterprise Edition technologies and enables service-oriented architectures by providing uniform data access for a wide variety of service and resource types. Not only does SDO enable a consistent approach to data access, but it also provides features that simplify common application tasks, such as allowing data browsing and update while the application is disconnected from the data source.

In this article we explain how SDO works and where it may fit into your own applications. We also take a closer look at how you can use SDO to retrieve and modify data stored in a relational database.

Kevin Williams is a software developer and is leading the effort to incorporate SDO technology into the WebSphere Application Server for IBM.

kjwillia@us.ibm.com

Brent Daniel is a software developer with IBM. He currently works on a JDBC data mediator service for WebSphere Application Server.

bhdaniel@us.ibm.com

SDO Concepts

At the core of SDO are DataGraphs and Data Mediator Services (DMS). In simple terms, an application program obtains a DataGraph from a DMS that is specific to some back-end data store. The program can then examine and/or update the data contained in the graph. Optionally the program can employ a DMS to propagate the entire set of updates back to the original data source.

DataGraph

As a program object, a DataGraph provides an in-memory, nonpersistent copy of data. A program can work with this data even when there's no connection to the original data source. The data in a DataGraph is organized as a group of DataObjects, which may be linked together (i.e., the data is structured as a "graph"). A DataGraph also contains a schema that describes the structure of the DataObject type(s) contained in the DataGraph. To handle updates, a DataGraph also maintains a change history to track all modifications made to the graph.

Each DataObject contains a set of properties, which can be primitive values or references to other DataObjects contained in the DataGraph. If a DataObject's schema is known at development time, a developer can use automated tools to generate typed interfaces that simplify DataObject access. Alternatively, the application can define the schema at runtime, allowing dynamic access of DataObjects. With either static or dynamic access, linked data can be accessed in either a breadth-first or depth-first manner. For example, if a DataGraph contains customers and related orders, then orders can be obtained directly from the DataGraph or from their parent customer DataObject. SDO also allows for accessing data through XML Path Language (XPath) subset expressions.

Disconnected Programming Model

The DataGraph's disconnected, data source–independent nature provides a simple programming model, supports common application patterns, and offers a potential performance advantage.

Today, J2EE application developers have a wide variety of persistence frameworks to choose from, such as JDBC, EJB, or JDO. These frameworks have different APIs and are often complex, requiring developers to spend a great deal of time learning multiple APIs rather than developing applications. Since SDO provides a single data access API regardless of the persistence mechanism, developers can choose the framework that best fits an application without using different APIs.

Some developers strive for similar independence by developing custom Java objects to encapsulate data from different data sources. This tactic makes use of the Data Access Object design pattern. SDO inherently supports this pattern, freeing developers from the need to develop their own infrastructure.

To improve performance, some types of applications can exploit the DataGraph's support for applying multiple updates in one method call to reduce the number of connections and/or database operations. By storing data from multiple database rows and tables in a DataGraph, applications can make changes to the data without making additional round-trips to the database.

Mediators

An SDO DataGraph must be populated from a data source or a service. The SDO component that populates a DataGraph is called a data mediator service (DMS). A DMS also propagates changes to the in-memory DataGraph back to the originating data source. Note that the current SDO specification does not define a specific API for DMSs – beyond a few basic requirements, each DMS provider is free to design the DMS that best suits the associated data source.

Typically, a DMS accesses a single type of data source, for example, JDBC resources or entity EJBs. All DMSs require the developer to provide a description of the data to be accessed. This data description (or metadata) typically consists of a schema and a query over the associated data source.

Figure 1, from the SDO specification, illustrates the flow of data during a typical interaction between an SDO client and a DMS. The client makes a request to the DMS to return a DataGraph. The DMS reads the requested data from the data source, constructs a DataGraph of related DataObjects, and returns the DataGraph to the application. The SDO client makes changes to the DataGraph in-memory and then sends the modified DataGraph back to the DMS. The DMS examines the ChangeSummary contained in the graph and propagates the changes back to the original data source.

Now your ideas can get even bigger.

Visual Studio .NET 2003 significantly reduces the amount of code you need to write, freeing you up to think big thoughts.

Think big. Visual Studio® .NET 2003 delivers higher productivity by taking care of the small stuff, so you can concentrate on bigger things. Just ask HP. "On our consumer application project Visual Studio .NET really lowered the technical barriers to doing great work," says an HP Technical Lead. "We developed the application faster than we could have with other technologies due to the better tool set capabilities." To find out how Visual Studio .NET 2003 can help you concentrate on big ideas, visit msdn.microsoft.com/visual/think

Microsoft®
Visual Studio®

Because the DataGraph is disconnected from the data source, it's possible that another application will update the data (in the data source) that was used to populate a DataGraph before the application requests the DMS to propagate the application's changes back to the data source. To handle such potential update conflicts, a DMS typically implements some form of "optimistic" concurrency control and throws an exception to the application when a data collision occurs. At that point, it is the application's responsibility to recover from the collision, for example, by rereading the data and restarting the transaction.

Too-frequent collisions under an optimistic concurrency approach can degrade performance as well as aggravate end users. In applications where multiple applications will often attempt concurrent changes to the same data, optimistic concurrency control may not be a good choice. However, for applications without this behavior, optimistic concurrency control can improve performance by reducing lock contention.

Metamodel

The SDO specification assumes the presence of a metamodel and metadata API for the DataGraph, but does not specify one explicitly. Today, SDO could be implemented with a variety of metamodels and schema languages such as XML Schema or the Essential Meta Object Facility (EMOF). The metamodel implementation does not affect SDO end users.

XML Serialization

SDO defines the XML format for DataGraphs and DataObjects, and specifies that the format can be customized by an XSD. This same format is used for Java serialization. The serialized form of a DataGraph includes the DataObjects as well as the schema and change summary. This capability allows data to be easily transferred over the wire as would be required by a Web service invocation.

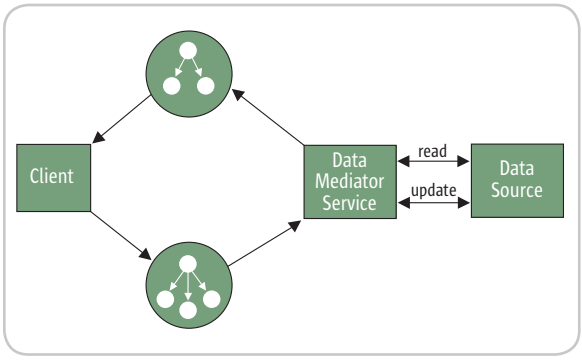


Figure 1

Relationship to Other J2EE Technologies

SDO can complement or simplify existing J2EE technologies. SDO complements JDBC by providing a more powerful framework and API for data access. In a typical relational database, data is normalized into multiple tables. When this data is read using a join query through JDBC, it's returned to the application in a tabular format that includes some data redundantly (e.g., an order number may be repeated with all individual line items for the same order). This format doesn't directly correspond to Java's object-oriented data model and can complicate navigation and update operations. A JDBC DMS can restructure this tabular data into a graph of related DataObjects. For example, an order might be represented by a DataObject that contains a list of references to other DataObjects containing line-item data. This allows an application to use standard Java techniques to access and modify the data.

Data access via EJBs can also be enhanced by using SDO. To implement a disconnected Data Access Object design pattern with EJBs alone, an application must use some combination of copy helper objects, session beans, and EJB access beans. An EJB DMS provides a ready-to-use disconnected architecture and frees developers from having to implement their own framework or custom artifacts.

SDO could also be used to complement other related technologies. For example:

- ***JSR 227:*** Declaratively binding and accessing data in J2EE applications. SDO could be used as the mechanism to return results from the data source.
- ***JSR 225:*** XQuery API for Java (XQJ). A Data Mediator Service could use the provided API to return SDOs.
- ***JDO 2.0:*** SDO could provide data transfer objects from JDO persistent objects.
- ***WEB UI Data Binding:*** JSF can use SDOs as a data binding. JSTL can use an SDO DataObject impl that implements the map interface.

Security

Security is not part of the current SDO model, so security in an SDO-based application is provided at the edges. For example, if an SDO-based application is developed that employs an EJB session bean and a JDBC connection, then security is provided at the boundaries of the application by these two J2EE components.

SDO with JDBC

SDO provides a uniform model for accessing data from a variety of services or data sources, including JDBC. Figure 2 shows interactions among the main artifacts involved when an application uses SDO over JDBC. Notice how the application code calls mediator and DataGraph methods, while the mediator calls JDBC and DataGraph methods, thus insulating the application from JDBC.

There are three central aspects to using a JDBC mediator: metadata, connections, and transaction handling.

Metadata

The application must supply the JDBC DMS with a metadata object that specifies the data to be retrieved. For a JDBC mediator, the metadata contains an augmented relational database schema that defines a set of tables, their relationships, and selection and ordering criteria. The JDBC DMS creates a DataGraph that corresponds to the provided schema definition. Each DataObject type within the DataGraph corresponds to a table definition in the schema, and each DataObject property corresponds to a table column.

The JDBC DMS uses the metadata to generate a SQL Select statement to retrieve data for the DataGraph. The simplest metadata example would describe a single table and no selection or ordering. For this specification, the JDBC mediator would retrieve all rows of the table and create a DataGraph that contains a list of DataObjects, with each

Altova® Authentic® 2005 – The powerful, FREE XML and database content editor.

ALTOVA® authentic® 2005

Recognize real value

Check out Authentic® 2005, the genuinely FREE XML and database content editor from Altova®.

Detected in Version 2005:

- Relational database content editing
- XPath 2.0 support
- XSLT 2.0 support

Put Authentic 2005 into action as the non-technical data entry element of advanced XML-based document frameworks and management oriented databases. Deploy Authentic 2005 to enable business personnel to edit content without being exposed to the underlying technology. They simply key in electronic forms using the word-processor style interface and their data is instantly adapted behind the scenes. Get your technology undercover!

Download Authentic® 2005 today:
www.altova.com

ALTOVA®

DataObject containing the data from one row in the table. Each DataObject will have a set of values corresponding to the values from each column.

A more complex example might involve two related tables; say Customers and their respective Orders. In this case, the metadata must specify the relationship between the two tables, which will subsequently be reflected in a corresponding relationship between two types of DataObjects.

The DataGraph returned in this case would contain a list of Customer DataObjects and each of these Customer DataObjects would have a list of related Order DataObjects. The DataGraph will contain all Customers and Orders; they are organized as a tree with Customers at the "root" of the tree and related Orders branching off of each Customer.

Applications will frequently want data only from specified rows of a table. In this case, the metadata for a JDBC DMS specifies selection criteria. For example, customers might be selected from a particular zip code or with a particular last name. Also, the DataObjects in the graph can optionally be ordered by specifying "order by" columns in the metadata.

Normally the JDBC DMS generates SQL select, insert, update, and delete statements to read and update the associated relational database. However, an application can optionally provide an explicit Select statement for the mediator to use. If this option is used, the DMS will then generate only the complementary insert, update, and delete statements and will use the provided select statement as is.

Connections

In addition to specifying what data to retrieve, an application must specify which data source the DMS should access. For a JDBC DMS, this can be done by specifying a JDBC Connection object. The DMS will use this connection for all database interactions.

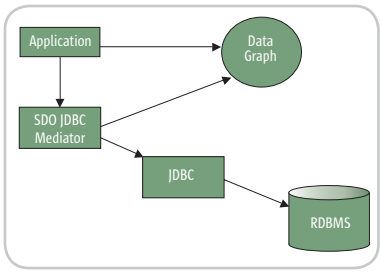


Figure 2

Transactions

As mentioned earlier, SDO provides a disconnected programming model and, accordingly, DMSs will typically connect to a data store only to read data for graph creation or to write data to reflect changes back to the store.

When an application requests the JDBC DMS to retrieve data and produce a DataGraph, the DMS starts a transaction, reads the data, creates the graph, and ends the transaction. The DataGraph is returned to the application and is "disconnected" in the sense that it is not associated with any connection or transaction; there are no locks held on the data.

The client can now read data from the DataGraph and make changes to it while it is in memory and disconnected from the data source. All changes made to the graph are recorded by the DataGraph. At some point the client will want to push these changes back to the data source and call the JDBC DMS "applyChanges" API.

As part of the "applyChanges" function, the JDBC DMS will reflect to the data store all changes made to the graph as part of a single transaction; this is true whether there is a single change to the graph or an entire batch of changes.

The disconnected programming model generally implies the use of an optimistic concurrency control scheme to push changes back to the data store; this is the approach taken by the JDBC DMS.

When the DMS attempts to apply DataGraph changes back to the data store, each row being updated is checked to ensure it has not been modified since it was originally read. If no intervening modifications have taken place, the update proceeds. If a row has been modified since the data was read, a collision has occurred; the update transaction is rolled back and an exception is thrown to the client.

There is also an option to use the DMS within a larger transaction. If this option is used, the DMS will assume that the client is controlling the transaction and will not perform any commit or rollback operations.

An Example

The following simple example demonstrates JDBC database access with SDO employing the JDBC DMS. This example has six steps, each illustrated by a code snippet.

Step 1: *Create the JDBC mediator metadata instance*

Create the metadata instance to represent the Customer table. This example demonstrates the creation of the JDBC DMS metadata programmatically. This is an obvious candidate for tooling support. Remember that the JDBC DMS uses a simple mapping scheme whereby each table definition results in a DataObject type and each table column results in a DataObject type property (see Listing 1).

Step 2: *Create the DMS instance as in Listing 2*

Step 3: *Read the DataGraph from the database*

```
//Create the "lastName" argument for the
filter predicate
 DataObject arguments = mediator.getParam-
eterDataObject();
arguments.put("CUSTLASTNAME", "Pavick");
DataObject graph = mediator.
getGraph(arguments);
```

Step 4 : *Retrieve data from the graph*

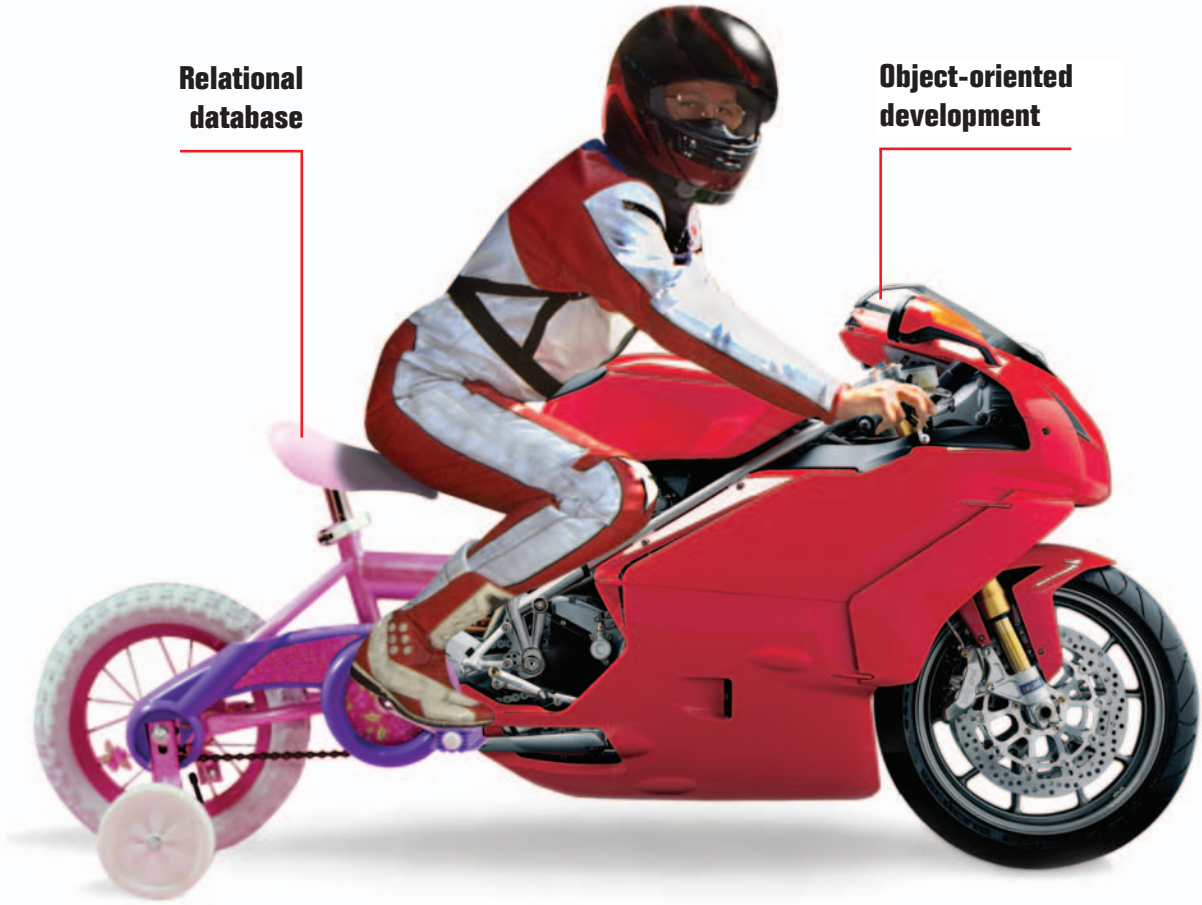
```
//Iterate through all returned customers
and print the first name
Iterator i = graph.getList("CUSTOMER").
iterator();
 while (i.hasNext()) {
 DataObject cust = (DataObject) i.next();
 System.out.println(cust.getString("CUSTFIR
STNAME"));
 }
```

Step 5: *Change the DataGraph*

```
List customers = graph.getList("CUSTOMER");
//Get the first customer in the graph and
update the name
DataObject customer =
(DataObject)customers.get(0);
 customer.setString("CUSTFIRSTNAME",
"Kevin");
```

Step 6: *Apply DataGraph changes back to the database*

```
mediator.applyChanges(graph);
```



A BETTER DATABASE CAN SPEED UP YOUR DEVELOPMENT CYCLE

If your back-end database isn't a good match for your front-end development, you need a new database.

Caché, the *post-relational* database from InterSystems, combines high-performance SQL for faster queries and an advanced object database for rapidly storing and accessing objects. With Caché, no mapping is required between object and relational views of data. Every Caché class can be automatically projected as Java classes or EJB components with bean-managed persistence. Plus, every object class is instantly accessible as tables via ODBC and JDBC.

That means huge savings in both development and processing time. Applications built on Caché are massively scalable and lightning fast. They require little or no database administration. And Caché's powerful Web application development environment dramatically reduces the time to build and modify applications.

We are InterSystems, a specialist in data management technology for over twenty-six years. We provide 24x7 support to four million users in 88 countries. Caché powers enterprise applications in healthcare, financial services, government, and many other sectors. Caché is available for Windows, OpenVMS, Linux, and major UNIX platforms – and it is deployed on systems ranging from two to over 10,000 simultaneous users.

InterSystems CACHÉ™
Make Applications Faster

Try a better database. For free.

Download a free, fully-functional, non-expiring version of Caché or request it on CD at www.InterSystems.com/match3

Variations on the Example

Metadata File

In Step 1 we created the mediator metadata programmatically. An alternative is to provide the metadata in the form of an XML file. Listing 3 is the XML representation of the Customer metadata.

Using this file, Step 1 would not be necessary and Step 2 would become:

Step 2: *Create the mediator instance as shown in Listing 4*

Static Types

The example provided above uses the dynamic access APIs of DataObject. The JDBC DMS also supports the use of static SDO types. To use the static API access to DataObjects, a set of static types is generated at development time and tools are provided for this purpose. Using static types provides a cleaner user API as well as a performance boost at runtime. The generation step is beyond the scope of this article, but this is what Step 4 looks like when using a static customer DataObject.

Step 4: *Retrieve data from the graph*

```
//Iterate through all returned customers
and print the first name
Iterator i = graph.getCustomers().itera-
tor();
 while (i.hasNext()) {
 Customer cust = (Customer) i.next();
 System.out.println(cust.getFirstName());
 }
```

Paging

The JDBC Data Mediator Service also provides a paging capability that can be useful for marching through large data sets. A pager interface provides a cursor-like next() capability. The next() function returns a graph representing the next page of data from the entire data set specified by the mediator metadata; a previous() function is also available. A CountingPager is also provided that allows the retrieval of a specified page from the data set.

Listing 5 illustrates paging through a large set of customer instances using a Counting Pager.

Conclusion

In this article we have explored some of the key SDO concepts and also drilled down into a specific use of the technology for relational database access employing a JDBC Data Mediator.

SDO is a standard from IBM and BEA and there is a reference implementation under development at www.eclipse.org/emf. This EMF-based implementation of SDO will also be delivered with WebSphere Application Server 6.0 and will be complemented by:

- JDBC Data Mediator Service
- EJB Data Mediator Service

It is anticipated that the 6.0 version of WebSphere Studio will contain complete support for creating applications that leverage SDO; this will include visual tooling to configure the JDBC DMS. With the power of SDO, relational data can be integrated into Web applications more easily than ever.

Acknowledgments

We would like to thank Stephen Brodsky and Tom Schneider for their assistance with this article.

Listing 1

```
//Create the basic schema description
//This describes the physical database table
//used to populate the datagraph
MetadataFactory mFactory = MetadataFactory.eINSTANCE.
Metadata metadata = mFactory.createMetadata();
 Table custTable = metadata.addTable("CUSTOMER");
 //The generated SQL SELECT will start from the single specified
 //root table
custTable.beRoot();
 Column custID = custTable.addIntegerColumn("CUSTOMERID");
 custID.setNullable(false);
 custTable.addStringColumn("CUSTFIRSTNAME");
 custTable.addStringColumn("CUSTLASTNAME");
 custTable.setPrimaryKey(custID);

 //Add a "lastname" filter to the Customer table descriptor
Filter filter = mFactory.createFilter();
 filter.setPredicate("CUSTOMER.CUSTLASTNAME = ?");
 FilterArgument arg = mFactory.createFilterArgument();
 arg.setName("CUSTLASTNAME");
 arg.setType(Column.STRING);
 filter.getFilterArguments().add(arg);
 custTable.setFilter(filter);
```

Listing 2

```
 //Prior to this point, the application must have
// acquired a JDBC connection for the mediator to use
 …

//Wrap the connection. The wrapper indicates whether or not the //
DMS will actively manage the transaction. The default factory
//method produces an "active" wrapper. Another method is //pro-
vided to create a "passive" wrapper which allows
//participation in a larger transaction scope
 ConnectionWrapperFactory factory =
   ConnectionWrapperFactory.soleInstance;
wrapper = factory.createConnectionWrapper(connection);

JDBCMediatorFactory fact = MediatorFactoryImpl.soleInstance;
 JDBCMediator mediator = fact.createMediator(metadata, wrapper);
```

Listing 3

```
<?xml version="1.0" encoding="ASCII"?>
<com.ibm.websphere.sdo.mediator.jdbc.metadata:Metadata xmi:ver-
sion="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:com.ibm.web-
sphere.sdo.mediator.jdbc.metadata="http:///com/ibm/websphere/sdo/
mediator/jdbc/metadata.ecore" rootTable="//@tables.0">
  <tables name="CUSTOMER">
    <primaryKey columns="//@tables.0/@columns.0"/>
    <columns name="CUSTOMERID"/>
    <columns name="CUSTFIRSTNAME" type="4" nullable="true"/>
    <columns name="CUSTLASTNAME" type="4" nullable="true"/>
    <filter predicate="CUSTOMER.CUSTLASTNAME = ?">
      <filterArguments name="CUSTLASTNAME" type="4"/>
    </filter>
  </tables>
</com.ibm.websphere.sdo.mediator.jdbc.metadata:Metadata>
```

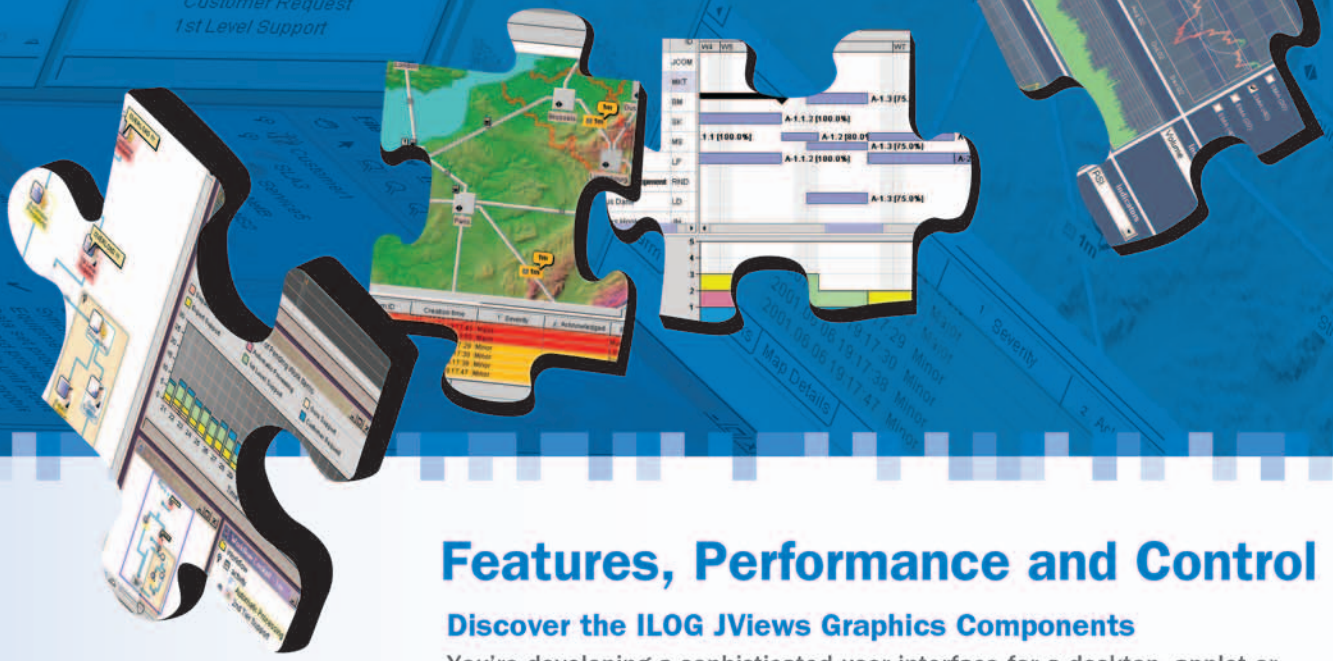
Listing 4

```
//Prior to this point, the application must have
// acquired a JDBC connection for the Mediator to use
 …

//Wrap the connection
 ConnectionWrapperFactory factory =
   ConnectionWrapperFactory.soleInstance;
wrapper = factory.createConnectionWrapper(connection);

InputStream stream = new FileInputStream("myMetadata.xmi");
JDBCMediatorFactory fact = MediatorFactoryImpl.soleInstance;
JDBCMediator mediator = fact.createMediator(stream, wrapper);
```

Listing 5

```
CountingPager pager =
PagerFactory.soleInstance.createCountingPager(5);

int count = pager.pageCount(mediator);
for (int pageNum = 1, pageNum <= count, pageNum++) {

DataObject graph = pager.page(pageNum, mediator);

//Iterate through all returned customers in this page
Iterator i = graph.getList("CUSTOMER").iterator();
 while (i.hasNext()) {
 DataObject cust = (DataObject) i.next();
 System.out.println(cust.getString("CUSTFIRSTNAME"));
 }

}
```

Get the complete picture



Features, Performance and Control

Discover the ILOG JViews Graphics Components

You're developing a sophisticated user interface for a desktop, applet or servlet application – it needs to provide displays that go far beyond what Swing and HTML offer. How can you be sure it will have the features, performance, customization and scalability to enable your end-users to make better more informed decisions, faster?

With ILOG JViews, you get comprehensive graphical libraries & tools, resources, and maintenance services so you can focus on the implementation, confidently completing your application in less time and at less cost.

Quickly and easily build:

- Gantt and resource displays
- Graph layouts, diagrams, workflows
- Geographic map displays
- Realtime data charts
- Custom monitoring and control screens
- Network and equipment management screens

Get a JViews Info Kit – Learn more, test drive an Eval.
Go to: jviews-info-kit.ilog.com or **Call: 1-800-for-ILOG**

ILOG®

Changing the rules of business™

Best Practices for JDBC Programming

Improving maintainability and code quality

by Derek C. Ashmore

While many new database persistence methods for Java programmers have been developed in recent years (e.g., entity beans, JDO, Hibernate, and many others), most database access code is still native JDBC. This statement doesn't express a preference, just an observation. Reasons for JDBC's popularity include: (1) it was first, (2) it works, and (3) most developers already know it.

I first documented "best practices" for using the JDBC libraries for *JDJ* in April 2000 (Vol. 5, issue 4). For the purposes of this article, the "best practices" goals for JDBC programming are *maintainability*, *portability*, and *performance*. *Maintainability* refers to the ease with which developers can understand, debug, and modify JDBC code that they didn't write. *Portability* refers to the ease with which JDBC code can be used with alternate databases. It turns out that JDBC does not make database programming as platform independent as I would like. In addition, I consider portability a noble goal even if you have no current plans to switch database vendors. Who knows how long your code will be around and what kinds of changes will have to be made to it? *Performance* refers to optimizing the time and/or memory needed to run JDBC code.

Derek C. Ashmore is a consultant and the author of the *J2EE Architect's Handbook*, available at www.dvtpress.com.

dashmore@dvt.com

Best Practices for JDBC Programming

Newer recommendations since my first article on the subject are the following.

Utilize Connection Pooling Techniques

Establishing database connections, depending upon platform, can take from 30 to 1,000 ms. This can be a meaningful amount of time for many applications if done frequently. Fortunately, all EJB containers and most servlet engines provide connection pooling features. Connection pooling provides a way for database connections to be established separately before your application needs them. They are then used and reused throughout the lifetime of the application. Furthermore, they're usually not difficult to configure and use. Listing 1 contains code that illustrates the use of connection pooling.

Listing 1 assumes the presence of a J2EE implementation. Applications without access to J2EE constructs can implement connection pooling features using one of many open source products. I recommend Commons-DBCP from Apache-Jakarta (http://jakarta.apache.org/commons/dbcp/). There is no need for developers to write their own connection pooling packages these days.

Connection pooling enhances performance by reducing the number of physical database connects and disconnects. Furthermore, it's common for connection pooling to have testing features that asynchronously test connections before your application needs them. In this way, pools provide applications with resilience to database outages.

It's very important that all connections created are closed. Note that with connection pooling, issuing a close() merely returns the connection to the pool. Usually, it doesn't result in a database disconnect.

Be Diligent About Closing All JDBC Objects

This practice is a reiteration of my previous article, but not closing JDBC objects after use is the most common mistake I see by far. Many developers don't understand that they are supposed to close ResultSet, Statement, PreparedStatement, and CallableStatement objects as well as Connection objects. Many are under the assumption that closing the connection will cascade into a close for these other types of objects. Some JDBC drivers do, but many don't.

Resource leaks caused by not closing JDBC objects are particularly aggravating because they may not surface until the code is run under load. In development, you may not generate enough resource leakage to cause a problem. Furthermore, some JDBC vendors override finalize() to release database resources after objects are garbage collected. If leaked objects are properly closed via finalize(), it's much harder to see the leak in development as the garbage collector corrects the leak. Under high load, JDBC objects may not be garbage collected soon enough to avoid exceeding database resources.

I guard against JDBC resource leaks by creating and closing JDBC objects within the same method. For example, the method that creates a connection will also be the method to close it. Furthermore, I close these objects in a finally block to ensure that they get closed in error conditions as well.

The reason many developers don't close JDBC objects in a finally block is that it's programmatically inconvenient. The fact that JDBC objects throw a checked exception on close() will cause you to encapsulate the closes in try/catch logic nested within the finally

block. An example of how to effectively close JDBC objects is presented in Listing 2.

To make closing a JDBC object more palatable, I usually create generic "close" utilities, like those I've included in the open source project CementJ (http://sourceforge.net/projects/cementj/), which make this close logic a one liner. For example, using a generic close utility, the close of the PreparedStatement and ResultSet objects in Listing 1 are one line:

```
DatabaseUtility.close(results, pStmt);
```

CementJ will check for nulls before attempting to close. It will also log exceptions received on the close, but not throw an exception (as there is typically nothing to be done anyway). Using a generic close utility, Listing 2 can be rewritten so that it's considerably shorter and less complex; Listing 2a illustrates this.

For those who wish to apply this concept but don't want to add an additional product dependency, Listing 3 illustrates an implementation for a generic close utility.

Check Stored Procedure Use

One of the most common questions I'm asked is if SQL should be embedded in stored procedures instead of the Java source. There's also a common perception that stored procedures always perform better than SQL statements executed directly within Java. Unfortunately, the answers to these questions aren't as simple as some would like.

There are really two things to consider with stored procedures: portability and performance. From a portability standpoint, stored procedures written in a proprietary language (such as Oracle's PL/SQL or DB2's SQL Procedure Language) make it much more difficult to migrate the application to another database should that become a business priority. These proprietary languages have unique features that might be difficult to replicate in other proprietary languages.

If you do write stored procedures, put them in a nonproprietary language. Some database vendors are supporting stored procedures written in Java and other third-generation languages. This makes them easier to move.

As to performance ramifications, the specifics can differ between databases. Stored procedures don't always perform better than embedded SQL. The soundest advice on this point is to comparatively measure stored procedure performance and embedded SQL for your database. As a general rule, CPU-intensive operations are bad candidates for stored procedures. For most types of databases, most of the performance gain comes from reducing the number of network transmissions, not from being parsed and physically stored in the database.

When considering whether or not to make something a stored procedure, I ask the question: How many network transmissions will be saved by making this process a stored procedure? If the answer is zero, performance will most likely not be improved.

Utilize Generated Key Value Retention Feature

One of the classic problems in database programming is how to handle generated unique key values. Most database vendors supply ways to automatically generate the key value and make it available when new rows are inserted. However, if an application needs that key value for other processing, it would have to read the row just inserted to get the generated value. For example, say purchase order numbers are dynamically generated by the database. A purchasing application would need the generated order number to put on the line items associated with that order.

The JDBC 3.0 specifies that the JDBC driver should return generated values with the insert operation. This means that you don't have to code and execute an additional select statement to retrieve the key value. Use of this feature can streamline code, but may or may not reduce the overhead of issuing an additional select depending upon the specific JDBC driver implementation. A word of caution, some database vendors do not yet support this feature of the JDBC specification.

To utilize the generated key feature using JDBC, execute a SQL statement that performs an insert. After execution, issue a getGeneratedKeys() from the Statement or PreparedStatement. The return will be a ResultSet with all generated keys (see Listing 4).

Separate JDBC Code from Business Logic

This is a design-level practice as opposed to a coding recommendation. I usually separate JDBC code into separate classes I call data access objects (DAOs). Data access objects manage access to relational databases as well as other types of persistent storage.

For convenience, I separate DAO objects in the package hierarchy (e.g., com.acme.appname.data or com.acme.appname.dao). Some developers also add a DAO suffix to data access object names; for example, a customer DAO might be named CustomerDAO.

The primary reasons to separate data access from the rest of the application is that it's easier to switch data sources and it's easier to share DAOs between functions or applications. Medium- to large-sized businesses in particular are likely to have multiple applications using the same data access logic. For example, it's common for an application to need customer information in several different functions. The act of logging into a Web site is different than e-mailing a purchase receipt. However, both these processes need customer information. Separating the access makes the access easier to reuse.

Since JDBC code is usually in separate packages, it's easier to locate should you wish to change database vendors or support multiple database vendors. As data access code is easier to find, the separation allows you to more easily determine the impact of database structure changes as well.

Consider Query Fetch Size for Large Result Sets

This practice is aimed at improving performance. The fetch size is the number of rows physically retrieved from the database at one time by the JDBC driver as you scroll through a query ResultSet with next(). For example, you set the query fetch size to 100. When you retrieve the first row, the JDBC driver retrieves the first 100 rows (or all of them if fewer than 100 rows satisfy the query). When you retrieve the second row, the JDBC driver merely returns the row from local memory – it doesn't have to retrieve that row from the database. This feature improves performance by reducing the number of calls (which are frequently network transmissions) to the database.

To set the query fetch size, set the fetchSize field on the Statement (or PreparedStatement or CallableState-

ment) before execution. Listing 5 provides an example of setting the query fetch size. (Listings 5–6 can be downloaded from www.sys-con.com/java/sourcec.cfm.)

As a general rule, setting the query fetch size is only effective for large result sets. If you set the fetch size much larger than the number of rows retrieved, it's possible that you'll get a performance decrease, not increase. Furthermore, the benefit you get from increasing the fetch size diminishes the higher it's set. I typically set this value to 100 for large result sets.

The performance improvement gained from setting the query fetch size varies widely depending upon the database platform and JDBC driver being used. I've seen performance improvements as large as 50%. Performance increases vary depending upon the speed of the network. Generally, the slower the network, the more performance increases can be gained by manipulating the fetch size.

Consider Update Batching

This practice is aimed at improving performance. In situations where you want to issue several inserts, updates, or deletes in the same unit of work, update batching allows you to group those statements together and transmit them to the database as one set of instructions. Like setting the query fetch size, update batching works by reducing the number of network transmissions between the application and the database.

For example, consider a Web site for online sales. When customers create orders, they often order multiple items. When the order is recorded, usually the items on the order are recorded at the same time. Update batching allows the multiple inserts for the order to be transmitted to the database at once.

Update batching is supported for SQL issued via PreparedStatement, CallableStatement, and Statement objects. An example of update batching is presented in Listing 6.

As with manipulating the query fetch size, the amount of performance improvement with batching statements varies between database vendors. I've seen performance improvements as large as 92% from batching statements. Also, the slower the network, the greater the opportunity for performance improvement.

Changes on the Horizon

The largest advance in the JDBC 3.0 specification is the addition of RowSet support. RowSets are ResultSets that eliminate the need for you to explicitly declare and use Statements and PreparedStatements for SQL queries. They were explicitly added to support the JavaBean specification. The supporting interfaces for RowSets are being included in Tiger. Optionally, you can download the JDBC RowSet Implementations 1.0 JWSDP 1.4 Co-Bundle 1.0 for use with v1.4 of the JDK.

RowSets can be connected or disconnected. Connected RowSets maintain an underlying connection to the database. Disconnected RowSets allow query results to be serialized and transmitted between JVMs, possibly on different servers. An interesting feature of disconnected RowSets is that you can update the RowSet and post those changes to the originating database at a later time. Another interesting feature is that RowSets can be serialized as XML documents.

As this is a new feature, providing "best practices" regarding the use of RowSets is premature. As with support for the generated keys feature, I would expect support for the RowSet feature to differ among the database vendors.

Summary

We've discussed several ways to make JDBC code more performant, maintainable, and portable on an individual basis. I always recommend team code reviews and documented coding standards as ways to develop more "best practices" and consistently apply existing coding techniques. Furthermore, team code reviews help further the goals of "best practices" by improving the maintainability and general quality of code within an application.

References

- *JDBC Technology Page:* http://java.sun.com/products/jdbc/
- *Commons DBCP – Connection Pooling package:* http://jakarta.apache.org/commons/dbcp/
- *JDBC RowSet Implementation Download:* http://java.sun.com/products/jdbc/download.html
- *JDBC Performance Tips:* www.javaperformancetuning.com/tips/jdbc.shtml
- *The J2EE Architect's Handbook:* www.dvtpress.com/javaarch

Core Best Practices

The practices recommended in my April, 2000 article were the following:

- *Use host variables for literals – avoid hardcoding them:* This practice involves using the java.sql.PreparedStatement instead of java.sql.Statement in cases where you need to supply values for "where" clauses in SQL statements. This eliminates database overhead in figuring out how to access your data. One noted exception is Oracle. Oracle, in recent releases, has tuned Statement processing to the point that it slightly outperforms PreparedStatement processing for small numbers of statement executions.
- *Always close Statements, PreparedStatements, and connections:* This practice involves always closing JDBC objects in a finally block to avoid resource limitations found in many databases.
- *Consolidate formation of SQL statement strings:* This practice involves placing the SQL statement text in a field that is declared static final to reduce string processing as well as make SQL statements easy to identify and read.
- *Use delegate model for a database connection:* This practice involves consolidating any database-specific tuning in a custom implementation of connection so that it's possible to take advantage of database-specific tuning features without sacrificing too much in the way of portability.
- *Use Date, Time, and Timestamp objects as host variables for temporal fields (avoid using strings):* This practice eliminates conversion overhead in the database and often the application.
- *Limit use of column functions:* This practice makes it easier to switch database vendors.
- *Always specify a column list with a select statement (avoid "select *"):* This practice insulates your code against tuning activities of database administrators.
- *Always specify a column list with an insert statement:* This practice insulates your code against tuning activities of database administrators.

OCTOBER BUZZ

Jtest®

An Ounce of Prevention

To maximize the benefits of coding standard analysis, it's critical that you recognize coding standards as a means of preventing errors—not detecting them. Many developers are disappointed if a coding standard violation doesn't point them to an obvious bug. When they explore a violation and find an error-prone construct rather than an error, they think that the coding standards aren't useful, eventually stop investigating violations, and later stop performing coding standard analysis altogether. This speaks to a fundamental problem with the software industry: Most development and testing teams are concerned with *removing* errors, but not with *preventing* them.

Error prevention involves correlating each error to the exact point in the development process that allowed it, then fixing that part of the process. This prevents the need to debug applications after the fact, and produces an exponential increase in product quality. Error prevention is very different than error detection, which is the process of finding and fixing errors *after* an application is built. When development focuses only on error detection, the flawed process that generated those errors is left uncorrected and the errors continue to occur. Furthermore, this misplaced focus has a negative impact on software quality and team productivity.

The coding standards promoted by industry experts were produced in response to other developers' mistakes and development process flaws. The experts performed the most difficult task: figuring out why errors were occurring and how to prevent them. To prevent errors, you just follow the coding standards that the experts designed and tested. Essentially, you get to benefit from the lessons learned from other developers' mistakes without having to suffer the consequences of personally committing those mistakes.

***– Adam Kolawa**, Ph.D.*
Chairman/CEO of Parasoft

Automate unit test case generation for JUnit and Java with Parasoft Jtest®

Introducing Parasoft® Jtest®

Parasoft Jtest is the first and only automated unit testing product for Java development.

With just a click, Jtest reads, analyzes and tests code — creating harnesses, stubs and inputs automatically. Jtest also enables you to automate regression testing and coding standard analysis. For loyal JUnit users, Jtest is designed to fully support existing test cases and automate the creation of new JUnit-style test cases.

Learn how Jtest can enhance JUnit capabilities...

Download a free eval copy of Jtest along with our informative new white paper entitled "Automated Java Unit Testing, Coding Standard Compliance, and Team-Wide Error Prevention."

For Downloads go to www.parasoft.com/jtest

Email: jtest@parasoft.com or Call: 888-305-0041 x3303

PARASOFT®
We make software work.™

FEATURES

- Fully integrated with JUnit and Eclipse
- Finds and fixes errors fast
- Automatically generates JUnit test cases
- Customizable testing and reporting

BENEFITS

- Makes error prevention feasible and painless, which brings tremendous quality improvements, cost savings, and productivity increases
- Makes unit testing and coding standard compliance feasible and painless
- Encourages the team to collaborate on error prevention

PLATFORMS

- **Windows 2000/XP**
- **Linux**
- **Solaris**

Contact Info:

Parasoft Corporation
101 E. Huntington Dr., 2nd Flr.,
Monrovia, CA 91016
www.parasoft.com

Recommendations

Listing 1

```
public void myMethod()
 {
  Connection conn = null;

  try
  {
   Context initContext = new InitialContext();
   Context envContext  =
    (Context)initContext.lookup("java:/comp/env");

   DataSource ds =
    (DataSource)envContext.lookup("jdbc/ora10g");
   conn = ds.getConnection();

   // Your application code here
  }
  catch (Throwable t)
  {
   // Your error handling code here
   t.printStackTrace();
  }
  finally
  {
   DatabaseUtility.close(conn);
  }
 }
```

Listing 2

```
    public PurchaseOrderVO getPurchaseOrder(int orderNbr)
    throws SQLException
  {
    PurchaseOrderVO order = null;

    PreparedStatement pStmt = null;
    ResultSet results = null;

    try
    {
// Application code here.
    }
    finally
    {
          if (results != null)
          {
                    try {results.close();}
                    catch (SQLException s)
                    {
                              // Log warning here.
                    }
          }
 if (pStmt != null)
 {
  try {pStmt.close();}
  catch (SQLException s)
  {
   // Log warning here.
  }
 }
 // Connection purposely not closed –
 //    managed elsewhere.
    }

    return order;
  }
```

Listing 2a

```
public PurchaseOrderVO getPurchaseOrder(int orderNbr)
    throws SQLException
  {
    PurchaseOrderVO order = null;

    PreparedStatement pStmt = null;
    ResultSet results = null;

    try
    {
// Application code here.
    }
    finally
    {
    // CementJ alternative for close
          DatabaseUtility.close(results, pStmt);
    }

    return order;
  }
```

Listing 3

```
public class DatabaseUtility
{
  public static void close(PreparedStatement pStmt)
  {
    if (pStmt == null) return;
    try  {pStmt.close();}
    catch (SQLException e)
    {
      LogManager.getLogger().logWarning("Prepared statement close
error", e);
    }
  }

  public static void close(Statement stmt)
  {
    if (stmt == null) return;
    try  {stmt.close();}
    catch (SQLException e)
    {
       LogManager.getLogger().logWarning("Statement close error",
e);
    }
  }

  public static void close(ResultSet rs)
  {
    if (rs == null) return;
    try  {rs.close();}
    catch (SQLException e)
    {
       LogManager.getLogger().logWarning("ResultSet close error",
e);
    }
  }

    public static void close(Object dbObj)
    {
      if (dbObj == null) return;
      if (dbObj instanceof PreparedStatement) close(
(PreparedStatement) dbObj);
      else if (dbObj instanceof Statement) close( (Statement)
dbObj);
      else if (dbObj instanceof ResultSet) close( (ResultSet)
dbObj);
      else if (dbObj instanceof CallableStatement) close(
(CallableStatement) dbObj);
      else if (dbObj instanceof Connection) close( (Connection)
dbObj);
      else
  throw new IllegalArgumentException(
"Close attempted on unrecognized Database Object!");
    }
}
```

Listing 4

```
private static final String INSERT_STMT =
  "insert into PURCHASE_ORDER (CUSTOMER_ID) values (?)";
 protected void runTest() throws Exception
 {
  PreparedStatement pStmt = null;
  ResultSet genKeys = null;
  try
  {
   pStmt = _dbConnection.prepareStatement(INSERT_STMT);
   pStmt.setString(1, "foo");
   pStmt.executeUpdate();

   // Generated Key Processing
   genKeys = pStmt.getGeneratedKeys();
   while (genKeys.next())
   {
    System.out.println(genKeys.getString(1));
   }
  }
  finally
  {
   DatabaseUtility.close(genKeys, pStmt);
  }
 }
```

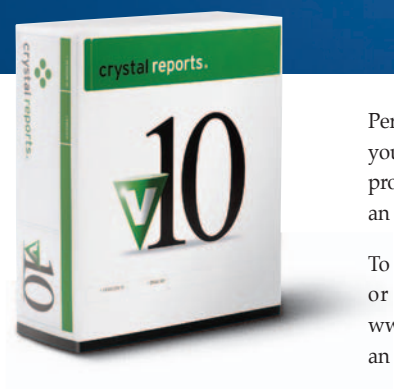
We'd like to think that not all perfect matches are made in heaven.

Crystal Reports 10

Which edition of Crystal Reports® is right for you?

	Report Author/IT Editions		Bundled Developer Editions		Full Developer Editions	
	Standard	Professional	.NET Edition [2]	Java™ Edition [3]	Developer	Advanced
Report Creation						
Visual report designer for rapid data access and formatting	●	●	● [1]	● [1]	●	●
Customizable templates for faster, more consistent formatting	●	●			●	●
Repository for reuse of common report objects across multiple reports [4]		●			●	●
Data Access						
PC-based and Microsoft® ODBC/OLE DB for MS Access and SQL Server	●	●	●	●	●	●
Enterprise database servers (ODBC, native)		●	● [1]	● [1]	●	●
Custom, user-defined data through JavaBeans™				●	●	●
Custom, user-defined data through ADO and .NET			●		●	●
Report Integration						
Report viewing APIs (.NET and COM SDKs)			●		●	●
Report viewing APIs (Java SDK)				●	●	●
Extensive report viewer options (DHTML, ActiveX, Java Plug-in, and more)					●	●
APIs for run-time report creation and modification						●
Report Parts for embedding report objects in wireless and portal apps	●	●			●	●
Report Deployment						
Crystal Reports components for report viewing, printing, and exporting:						
a) Java reporting component				●	●	●
b) .NET reporting component			●		●	●
c) COM reporting component					●	●
Full featured report exporting		●			●	●
Report server (Crystal Enterprise Embedded deployment license)						●

1 Limited functionality. 2 Bundled with Microsoft® Visual Studio® .NET and Boland® C#Builder™.
3 Bundled with BEA WebLogic Workshop™ and Boland® JBuilder®. 4 This feature is available on the Crystal Enterprise CD, included in the Crystal Reports 10 package.



Perfect matches can be made here too. In order to quickly determine which Crystal Reports® best suits your project requirements, we've provided this basic feature chart. Crystal Reports® 10 simplifies the process of accessing, formatting, and tightly integrating data into Windows and web applications via an enhanced designer, flexible data connectivity options, and rich Java™, .NET, and COM SDKs.

To learn more about Crystal Reports 10, compare over 150 different features across versions, or to access technical resources like the Developer Zone and evaluation downloads, visit: www.businessobjects.com/dev/p7. To ask more specific report project related questions, contact an account manager directly at 1-888-333-6007.

BUSINESS OBJECTS®

Integrating with Eclipse

Interview by Bill Dudney

An interview with Lee Nackman

Vice President, Desktop Development Tools & CTO, Rational Software

In July IBM announced that the Rational tool set would be fully integrated within the Eclipse tool set and would provide an integrated set of tools to support the full life cycle of software development. Recently I was able to interview Lee Nackman, the CTO of the Rational division of IBM.

In the press release IBM announced that the Rational tool set would fully embrace Eclipse. What does "fully embrace" mean?

The next release (due by year's end) will have the ability to address several different roles in the software development process – analysts, architects, developers, and even project managers. The development platform is an integrated tool set that supports each of these roles and allows them to work together toward a common goal. Eclipse becomes the key technology to integrate all the tools for each of these roles and for integrating all the capabilities that are needed across all these roles. For example, requirements affect each of the roles (analyst, architect, designer, developer, and tester), and quality affects the people playing each of these roles as well. The tools need to integrate very tightly to allow these people to work together, and Eclipse is the technology to integrate the tools. For example, we support the various roles with Eclipse's perspective mechanism, which allows users in different roles to have a different view of the tool than users in some other role.

We think it's important for the tools to share internal models rather than export, import, and transform, which has been the traditional integration point behind these tool sets in the past. Consider the Eclipse Modeling Framework (EMF): we use EMF and some of the standard metamodels, like the UML 2.0 model that is also provided by Eclipse, as a means to provide integration across the tool set.

The extensibility features of Eclipse are also very important for us. Our tools are open and have to be extensible both by our customers and our partners and other third parties. Eclipse is central to this whole notion of integration and extensibility.

It sounds like the model that the analyst is working with is the same model that, for example, the tester is working with; they just have a different view into the same information.

Exactly. Let me give you a concrete example. When a developer uses Eclipse to edit some source code, assume a method is added. The class in the UML model is automatically updated to reflect these changes. If instead of editing the code the developer added the method to the UML model, the method would be automatically added to the Java code. In essence the code and the model are basically views into the same information, the same model. This is already being done in our current product set.

What about the architect who does not want to model all the detail? Is he or she able to work in a disconnected mode, where the model is not directly tied to the code?

Yes, but one of the things that we think is very important is for the tools to have the ability to analyze the code and let the architect see the architecture as built, as opposed to merely specifying the architecture. With our tool set the architect can see what is actually built and then compare that with what was designed.

Architects don't necessarily want to see all the detail, however. What support does the tool set provide for a more abstract view of the application?

With our current product set you can reverse engineer an entire J2EE application and see an abstract view of the code. For example, instead of seeing the local interface, home interface, and implementation of an EJB entity, you would see that as a single UML class. You can then create (or view) associations between the entities that are converted to EJB relationships behind the scenes, but as the architect or developer you don't have to see all that detail (or complexity). We do this today in our shipping product and we will deliver more features like this in our next release.

Often an architect is concerned with an even more abstract view of the code. Is it possible to see the implementations patterns like Session Façade, for example, in the model?

Basically yes, but there is no magic here. You can take a big set of code and visualize that. You don't have to see all of it at once either. Various aspects of the application can be represented in different diagrams. The tool will show you the artifacts of your application in UML and allow you to see the relationships that exist. It's up to the developer, designer, and/or architect to discern if a particular pattern is correctly applied.

When you say "not all at once" do you mean that the tool will build multiple diagrams for you?

No, the tool will build a model of your application and the model will be rich in terms of including a lot of detail about the code, relationships, etc., but

Bill Dudney, *JDJ*'s Eclipse editor, is a senior consultant with Object Systems Group. He has been doing Java development since late 1996 after he downloaded his first copy of the JDK. Prior to OSG, Bill worked for InLine Software on the UML bridge that tied UML Models in Rational Rose and later XMI to the InLine suite of tools. Prior to getting hooked on Java, he built software on NeXTStep (precursor to Apple's OS X). Bill has roughly 15 years of distributed software development experience starting at NASA building software to manage the mass properties of the space shuttle. You can read his blog at http://jroller.com/page/BillDudney.

billdudney@sys-con.com

the tool will not automatically build the interesting diagrams. It's a manual process for the designer to draw the diagrams. The tool makes it simple though. Since the model exists, it's simply a matter of selecting which elements of the model should be shown on which diagram. Making things look nice still requires human intervention.

Back to the integration provided between the analyst and the tester. What support does the tool set provide to allow these two people to work together? For example, a tester might update a test case because a particular condition was incorrect. How does the tool support a change like this getting propagated back into the use case?

This is a two-part answer. The first part takes us all the way back to requirements. With our ReqPro product the analyst can drop a requirement onto the use case model. The tool will automatically create the use case and then connect the use case back to the requirement through a traceability link. Now let's move on to your specific question. In the same way that the requirements and use cases can be tied together with traceability links, there are ways to connect test cases with use cases. In addition you can connect the test cases with the results of running the tests. Then the project manager has insight into the current state of the project at any time. And the analyst and testers have insight into dependencies being changed. Now the integration is not as complete as we'd like it to be, but we are making good progress in this next release.

Moving on to the use of the Eclipse UML 2.0 model. To what extent are you using this model and to what extent is your team involved in providing feedback to Eclipse?

We are using the model and providing feedback to the team. We are continuing to invest very heavily in the combined IBM and Rational teams providing quality feedback to the Eclipse ecosystem.

How would you characterize your use of Eclipse? Has it been an advantage to start with the large base of tools already delivered with Eclipse?

It's working quite well for us. It's more expensive for us to work within the open source model because of the level of communication that's required to make sure that everyone is on board with whatever changes are being proposed and/or made. That expense is repaid because of the ecosystem that is developing around Eclipse. There is a large group of vendors building on top of Eclipse. This group helps to refine and flesh out the extensibility features. There is a huge user group that (1) provides usability feedback on how Eclipse can be improved and (2) if someone knows how to use Eclipse they already know a lot about how to use our tool set.

The excitement for us is not so much in saving engineering costs by using Eclipse but rather the ecosystem that is developing around Eclipse. This ecosystem together with the fact that the technology in Eclipse is so good is what makes building on top of Eclipse so exciting for us. We don't need to build a new Java parsing infrastructure but we also don't have to try to teach our customers how to use the output of that infrastructure nor do we have to invest in educating our partners in how the infrastructure works. It's all a part of Eclipse that is widely adopted and documented. We can concentrate on building additional capabilities on top of Eclipse for our customers.

> "The excitement for us is the ecosystem that is developing around Eclipse"

How does the use of Eclipse fit into your portability picture? Will your tool set be strictly reliant on the Eclipse platform so that they will run anywhere that Eclipse runs?

In the next release there has been a big focus on integrating our products with Eclipse. There are still some clients that will not work on all Eclipse-supported platforms. The idea going forward is to make our tools run on all platforms that are supported by Eclipse, however, there are still the practical issues of being able to test the products on each platform that will prevent us from supporting them on every platform.

On the Model Driven Architecture (MDA) front, where do you see Rational fitting into that market?

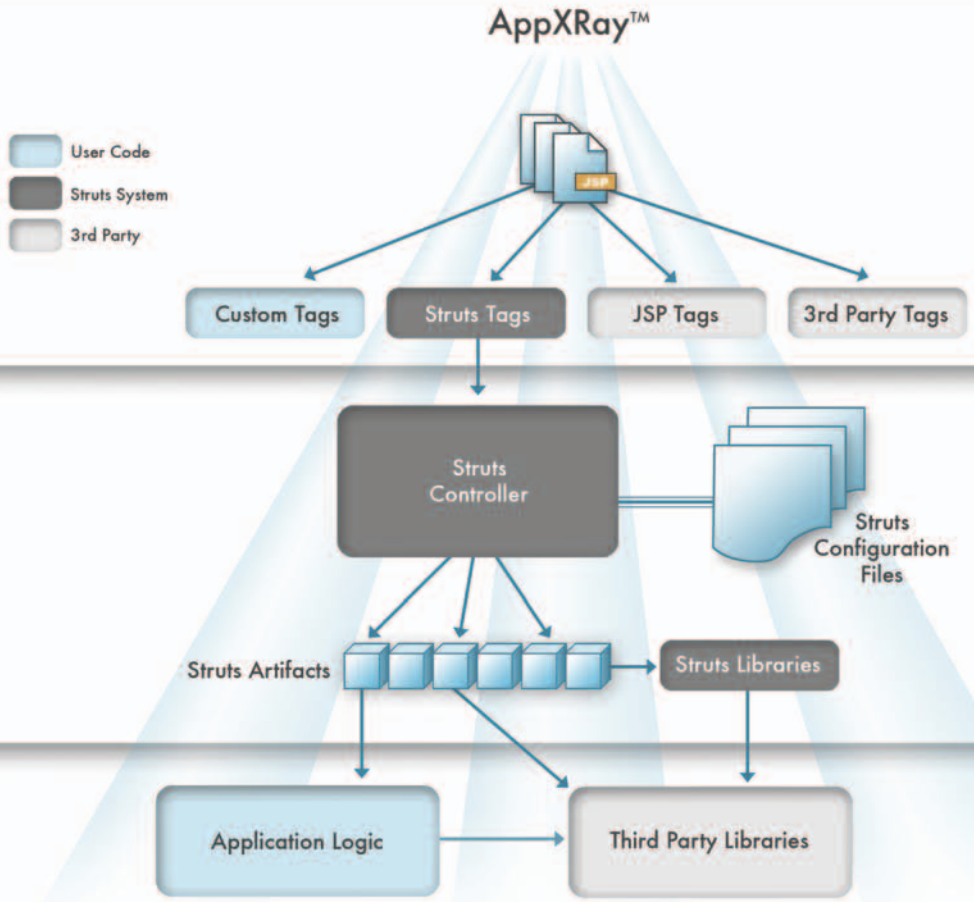
There are certainly markets where MDA will be valuable and we think there is opportunity and value in building models and generating code. For example, we've seen our customers generate a lot of value from defining data models and generate code from them. Take an XML Schema. Our customers have been able to achieve value in defining a model and having the tool build the XML Schema and database schema for them. User interfaces is another place where we have found modeling to be valuable. Having our users be able to draw out a user interface in a modeling tool and then have the tool generate all the code behind that UI is extremely valuable. In our current tools, for example, you can fully define your user interface in a visual editor and the tools will generate a JavaServer Faces implementation behind the scenes.

We don't believe in programming with pictures, however; there are things that are best represented in a textural format.

Finally how are you integrated with the Hyades project?

We provide integration with our testing tools so that you can deploy your J2EE application. With the monitoring pieces of Hyades in place you can see what is happening on the server; for example, if you have a search page that is having performance problems. With the Hyades integration the developer is provided with clues that will help nail down where this problem is happening and what needs to be fixed to make the problem go away.

Do you understand all the relationships in your web application?



Nitrox for JSP | Nitrox for Struts | Nitrox for JSF

NitroX™ AppXRay™ Does!

- JSP debugging including stepping in/out of JSP tags and unique JSP variables view
- Source and Visual JSP & Struts editors with knowledge of all web app layers
- Design time validation, consistency and error checking for JSP source,Tag Libs, Struts and Server-based Java code in a web app
- Eclipse & IBM WSAD, WSSD plug-in

Download a free, fully functional trial copy at: www.m7.com/d7.do

Managing Objects Between Java and C

Building the Java API on top of the native API

by Hari K. Gottipati

If you've ever used JNI, you know how to manage the primitive data types between Java and the native language. As you delve into JNI, particularly when developing a Java API on top of a native API, you need to know how to manage the objects between Java and the native language. This article takes you to the next level and walks you through the step-by-step process of managing the objects between Java and C.

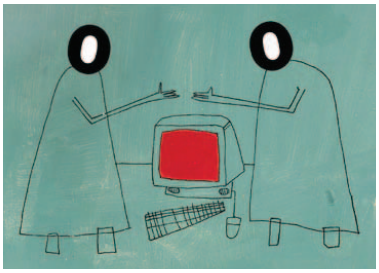
Why JNI

Platform independence and flexibility are the popular buzzwords in the industry and programmers want to prune their code. Everyone wants to program in their favorite language, unfortunately you can't always stick to your selection, as some components of an application have to be programmed in a completely different language or some components are only available in a different language. The solution to these situations is JNI. The Java Native Interface (JNI) is the native programming interface for Java that's part of the JDK. JNI allows Java code that runs within a Java Virtual Machine to operate with applications and libraries written in other languages, such as C or C++.

In addition, if you want to do some system-level programming with Java, you'll definitely end up using JNI somewhere. JNI is essential for bridging the gap between Java and the native functions. It allows you to write some code in a native language and then talk to that native code through Java.

Real-Life Scenario

When I wanted to develop a wireless mail solution for a particular mail server, I realized that Java was well suited for developing wireless solutions; however, that mail server had a C API, not a Java API. I then started writing the Java API using JNI on top of the C API. The main obstacle I faced was while passing the objects between Java and C. This article explains how I overcame this obstacle. By the end of this article you'll be familiar with the process of managing objects between Java and C. To understand this article you should be familiar with the basic concepts in JNI (see the Resources section for the JNI documentation).



Why Do You Need to Manage the Objects Between Java and C?

If you're developing the API, sometimes you may need to store an object in Java that you created in C; for example, a mail API. First log into the server with a username and password, then you get the session. Say you wanted to store that session in Java for later use, e.g., for reading or writing an e-mail. If you save the session in Java, you can then pass this session to C to read or write e-mail, etc.

Managing the Objects Between Java and C

You should already know how to pass the primitive types between Java and C (otherwise see the Resources section); it's straightforward. Unfortunately you can't directly convert the objects between Java and C like the primitives. You can indirectly store the C object memory location in Java through a long variable. Later, if you want to pass that object to C, simply pass the memory location. From the memory location you can retrieve the C object.

To maintain the same object in Java and C, create some object in Java and declare a long variable in it to hold the memory location of the object created in C.

I'll illustrate this in detail with the mail example. The first part explains how to fill the Java object from C and the second part explains how to get the C object out of the Java object (to help you understand, I bifurcated each section into a Java side and a C side).

Filling a Java Object from a C Object

Figure 1 shows the different steps involved in this process.

The following example discusses these steps in more detail:

- Declare a global long variable:

```
public long nativeSession;
```

- Declare a native method:

```
public void native jniLogin(String server,
String userName, String password);
```

- Define a Java API method to expose the native method:

```
public void login(String server, String
userName, String password) {
  jniLogin(server, userName, String pass-
word);
 }
```

Listing 1 provides a nativeSession variable, which will hold the session object that actually gets created in C through the native method jniLogin. Once we create the session object in C, we'll fill this nativeSession variable that's declared in Listing 1 with the memory location of the C object. I'll discuss this in-depth later.

Hari K. Gottipati is a software engineer at Aligo. He is involved in developing a mobile PIM solution for all known mail and calendar servers that can work on any kind of device. He is a seven year Java veteran specializing in mobile computing. Hari earned his master's degree in computer science from Bharathidasan University, India.

hari_gottipati@aligo.com

Javavavoom!

Introducing a high-performance database that's 100% Java.

Berkeley DB Java Edition

Download at www.sleepycat.com/bdbje

Finally there's a high-performance database that loves Java just as much as you do: Berkeley DB Java Edition (JE). Brought to you by the makers of the ubiquitous Berkeley DB, Berkeley DB JE has been written entirely in Java from the ground up and is tailor-made for today's demanding enterprise and service provider applications.

Java POWERED

Berkeley DB JE has a unique architecture that's built for speed. The software executes in the JVM of your application, with no runtime data translation or mapping required. Plus Berkeley DB JE has been specifically designed to handle highly concurrent transactions, comfortably managing gigabytes of data. And because it's built in your language of choice, your organization enjoys shorter development cycles and accelerated time-to-market.

Experience the outstanding performance of Berkeley DB JE for yourself.
Download Berkeley DB JE today at **www.sleepycat.com/bdbje**. Register now, and you'll also receive a 15% discount on a commercial license purchased before November 30, 2004.

> "Platform independence and flexibility are the popular buzzwords in the industry"

I won't explain the step about creating the header file out of a Java file using the javah command (see Resources for this information), so I'll jump directly to the C implementation of this native method.

Let's see the C implementation of the native method jniLogin:

- Create the native Object:

```
Session msession = NULL;
            connect(mserver, muserName,
mpassword, &msession);
```

- Get the memory location of the object:

```
jlong  sessionLocation = (jlong)msession;
```

- Get the Java object from the parameters:

```
jclass cls = (*env)->GetObjectClass(env,
obj);
```

- Fill the Java objects long variable with a memory location:

```
jfieldID fid =(*env)->GetFieldID(env, cls,
"nativeSession","J");
            (*env)->SetLongField(env,obj,f
id,sessionLocation);
```

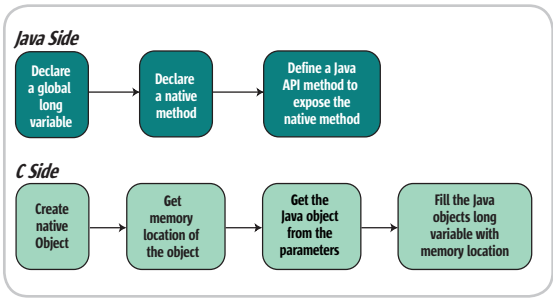


Figure 1 Filling a Java object from a C object

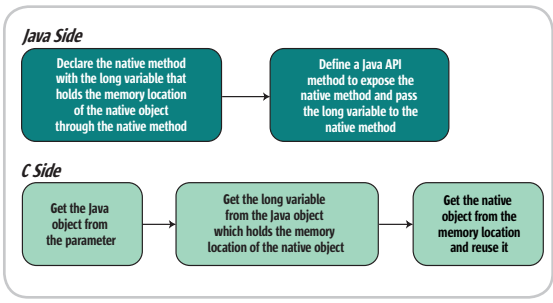


Figure 2 Getting the C object back out of the Java object

In line 8 of Listing 2, I'm declaring the Session(C) object. In line 14, to connect to a particular mail server, we're passing the values of the server, the user name, and the password to the connect method. If the given user name and password is valid, the msession object gets filled. Note that the function "Connect" is already defined in the C API of the mail server. Now we have the session object that logged into the server in C. In line 15 we're getting the memory location of the msession object. When you do (jlong)msession, it gives the memory location of the msession object. In line 20 we are getting the Java object that called this C function. In line 21, we're getting the long field(variable) "nativeSession" that we declared in Session(Java). In line 20 we're setting the long variable with the variable sessionLocation. Now the variable nativeSession of Session(Java) object holds the memory location of the msession(C) object.

Getting the C Object Back Out of the Java Object

Figure 2 explains the different steps involved in this process.

The following example provides a clearer picture:

- Declare the native method with the long variable that holds the memory location of the native object through the native method:

```
public void native jniSendMail(Object o,
String to, String subject, String body);
```

- Define a Java API method to expose the native method and pass the long variable to the native method:

```
public void sendMail(String to, String sub-
ject, String body) {
 jniSendMail(this, to, subject, body);
}
```

Let's say you wanted to send an e-mail with the session that logged in through the login method. For that we are passing the session object (with "this" keyword) to the jniSendMail method in Listing 3. Notice that session object has the nativeSession variable that holds the memory location of the native session object. With that we can send the e-mail. Let's see how we can get the native session back from this object in the next steps:

- Get the Java object from the parameters:

```
jclass cls = (*env)->GetObjectClass(env,
sessionobj);
```

- Get the long variable from the Java object that holds the memory location of the native object:

```
jfieldID fid =(*env)->GetFieldID(env, cls,
"nativeSession","J");
            jlong session = (*env)->GetLon
gField(env,obj,fid);
```

- Get the native object from that memory location and reuse it:

```
Session *  msession = (Session)session;
            sendMail(msession, mto, msub-
ject, mbody);
```

In line 14 of Listing 4, we're getting the Session(Java) class. In line 15, we're getting the variable nativeSession. Line 16 provides the value of the nativeSession variable that has the memory location of the native session. Line 18 has the C object from the memory location, since the long variable session holds the memory location of the native session object. When you typecast with (Session)session, it reads the object from the memory location. Now we have the C object. With this in line 19 we are calling the SendMail function with the values msession, mto, msubject, and mbody, which sends the e-mail to the address specified in the field "mto". Note that "SendMail" is the function that's already defined in the C API of the mail server.

Resources

- *Java Native Interface tutorial:* http://java.sun.com/docs/books/tutorial/native1.1/TOC.html
- *Java Native Interface Specification:* http://java.sun.com/j2se/1.4.2/docs/guide/jni/

Listing 1: A Java class with a long variable that will hold the native object's memory location

```
public class Session {

 public long nativeSession;

 public void native jniLogin(String server, String userName,
String password);

 public void login(String server, String userName, String pass-
word) {
   jniLogin(server, userName, String password);
 }

}
```

Listing 2: Native code that fills the long variable of a Session class with native objects memory location

```
1: JNIEXPORT void JNICALL Java_Session_jniLogin (JNIEnv * env, job-
ject obj, jstring server, jstring userName, jstring password) {
2: const char *mserver;
3: const char *muserName;
4: const char *mpassword;
5: jclass cls;
6: jfieldID fid;
7: jlong  sessionLocation;
8:          Session msession = NULL;
9:
10:        mserver = (*env)->GetStringUTFChars(env,server,0);
11:        muserName = (*env)->GetStringUTFChars(env,userName,0);
12:        mpassword = (*env)->GetStringUTFChars(env,password,0);
13:
14:        connect(mserver, muserName, mpassword, &msession);
15: sessionLocation  =  (jlong)msession;
16:        (*env)->ReleaseStringUTFChars(env, server, mserver);
17:        (*env)->ReleaseStringUTFChars(env, userName, muserName);
18:       (*env)->ReleaseStringUTFChars(env, password, mpassword);
19:
20:        cls = (*env)->GetObjectClass(env, obj);
21:        fid =(*env)->GetFieldID(env, cls,"nativeSession","J");
22:        (*env)->SetLongField(env,obj,fid,sessionLocation);
23: }
```

Listing 3: A Java class with another native method that uses the native session that gets created earlier

```
Public class Session {

 public long nativeSession;

 public void login(String server, String userName, String pass-
word) {
   jniLogin(server, userName, String password);
 }

 public void native jniLogin(String server, String userName,
String password);

 public void sendMail(String to, String subject, String body) {
   jniSendMail(this, to, subject, body);
}

public void native jniSendMail(Object o,String to, String subject,
String body);


}
```

Listing 4: Reusing the session object that is stored in the long variable of a session class

```
1: JNIEXPORT void JNICALL Java_Session_jniSendMail (JNIEnv * env,
 jobject obj, jobject sessionobj, jstring to, jstring subject,
 jstring body) {
2: const char * mto;
3: const char * msubject;
4: const char * mbody;
5: jclass cls;
6: jfieldID fid;
7: jlong session;
8: Session * msession = NULL;
9:
10:        mto =  (*env)->GetStringUTFChars(env, to, 0);
11:        msubject =  (*env)->GetStringUTFChars(env, subject, 0);
12:        mbody =  (*env)->GetStringUTFChars(env, body, 0);
13:
14: cls = (*env)->GetObjectClass(env, sessionobj);
15:        fid =(*env)->GetFieldID(env, cls,"nativeSession","J");
16:        session = (*env)->GetLongField(env,obj,fid);
17:
18:        msession = (Session)session;
19:        SendMail(msession, mto, msubject, mbody);
20:
21:        (*env)->ReleaseStringUTFChars(env, to, mto);
22:        (*env)->ReleaseStringUTFChars(env, subject, msubject);
23:        (*env)->ReleaseStringUTFChars(env, body, mbody);
24: }
```

Calvin Austin
Core and Internals Editor

Mastering Multithreading

Some of you may remember a time when the world of multithreaded programming was limited to a small set of C or C++ applications. Often the threads were used sparingly and restricted to a specific task or computation or even operating system.

When the Java platform arrived it brought with it the ability for anyone to write a fully multithreading program, including those with very little experience in using threads. Even the vendors who had the foresight to implement a threading library in the OS found they needed to fix those libraries in order to let these new multithreaded Java programs run.

The original Java threading model had very basic controls. Synchronized methods and blocks were there to guard code and a wait and notify pattern provided limited conditional support. Things were even more confusing when you had a thread start method but the suspend, resume, and stop counterparts were all deprecated! It wasn't too surprising that talks, books, and articles about using Java threads soon became very popular.

One book in particular stood out, partly due to the title but primarily because of the ideas it contained. That book was *Concurrent Programming in Java: Design Principles and Patterns* by Doug Lea (Addison-Wesley). This book was different because it looked at designing true concurrent, multithreaded programs first, and then applying that to the Java platform. In the process it uncovered areas where the Java APIs were limited. Instead of forcing developers to work around this restriction, Doug created the Java concurrency API library – a library that provided higher-level thread controls, a complement of locks, and finer-grained synchronization tools.

Fast forward to J2SE 5.0, which should be final by the time you read this. I'm pleased to report that the Java concurrency library is now part of the Java platform. Does that mean you should go back and rework every single piece of threaded code you wrote over the last eight years? Not necessarily. However, if you are starting from scratch or refactoring some historically tricky threading code, I would recommend spending a little time checking out what this library has to offer.

Along with synchronization controls like semaphores and explicit calls to create, set, and release locks, there is a significant upgrade to the thread controls. This new functionality is provided by the ThreadExecutor class. You're probably very aware that when you create a runnable task, there is no mechanism to return a result, throw an exception to the parent, or even cancel that task. Where before you may have used a Runnable task, you can now create a Callable one instead. They're very similar; the following is a small example to show how easy this is.

Here is my runnable task, called Worker. In the Callable version all I have to do is describe my return type, in this example a string, and use the new declaration of Callable that has one method named call instead of run. Notice that with Generics we can explicitly declare the return type so it can be checked at compile time.

```
class Worker implements Callable<String> {
    public String call() {
        System.out.println("New Worker");
        return "return result";
    }
}
```

To use my task I just need to pick an ExecutorService. In this example I chose the cached thread pool, which will create as many threads as I need but will reuse previously constructed threads if available. The logic behind this is that thread creation can be an expensive operation on some systems.

I create my Callable task, called task, and then create a handle to that task, a Future, that I have named result. The Future task serves two purposes. The first is that I can cancel the task by calling the method result.cancel(true). The true argument means that it's okay to try to cancel an already running task. The second benefit is that it's used to hold the return result. To gain access to the return result, I simply had to call the method result.get().

```
ExecutorService es = Executors.new-
CachedThreadPool();
Callable<String> task = new Worker();
Future<String> result;
result=es.submit (task);
try {
    System.out.println(result.get());
}catch(Exception e){}
```

This small example enabled me to use a cached thread pool to hand off tasks, check their status, and cancel them if required. As I mentioned earlier, I used an ExecutorService that declares some convenient default implementations. The concurrency library has a full set of interfaces and many other implementations that you can use. I encourage you to check the library out along with the other J2SE 5.0 features.

> "I'm pleased to report that the Java concurrency library is now part of the Java platform"

A co-editor of *JDJ* since June 2004, **Calvin Austin** is the J2SE 5.0 Specification Lead at Sun Microsystems. He has been with Java Software since 1996 and is the Specification Lead for JSR-176, which defines the J2SE 5.0 ("Tiger") release contents.

calvin.austin@sys-con.com

Java Programming: The Java Async IO Package

by Mike Edwards and Tim Ellison

Fast, scalable IO for sockets and files

The Async IO package is designed to provide fast and scalable input/output (IO) for Java applications using sockets and files. It provides an alternative to the original synchronous IO classes available in the java.io and java.net packages, where scalability is limited by the inherent "one thread per IO object" design. It also provides an alternative to the New IO package (java.nio), where performance and scalability are limited by the polling design of the select() method.

As its name implies, the Async IO package provides asynchronous IO operations, where the application requests an IO operation from the system, the operation is executed by the system asynchronously from the application, and the system then informs the application when the operation is complete. The Async IO package supports a number of styles of application programming and gives the application designer considerable freedom in the management of the number of threads used to handle IO operations and also in the design of the components that handle the asynchronous notifications.

Dr. Mike Edwards is a strategic planner in the IBM Java Technologies group in Hursley, England. He is responsible for technical planning for future IBM products including the IBM Java SDKs and for Web services–related products. Before working on Async IO, Mike was involved in the planning of Java SDK 1.4.0 and was a member of the Expert Group for JSR 059, which defined the specification for J2SE 1.4.0 and JSR 051, which created the New IO package. Mike received his PhD in Elementary Particle Physics from Birmingham University.

mike_edwards@uk.ibm.com

Why Java Applications Need the Async IO Package

The question "Why do Java applications need the Async IO package?" can be answered in two words: performance and scalability.

Performance and scalability are key attributes of the IO system for IO-intensive applications. IO-intensive applications are typically, although not exclusively, server-side applications. Server-side applications are characterized by the need to handle many network connections to many clients and also by the need to access

many files to serve requests from those clients. The existing standard Java facilities for handling network connections and files do not serve the needs of server-side applications adequately. The java.io and java.net packages provide synchronous IO capabilities, which require a one-thread-per-IO-connection style of design, which limits scalability since running thousands of threads on a server imposes significant overhead on the operating system. The New IO package, java.nio, addresses the scalability issue of the one-thread-per-IO-connection design, but the New IO select() mechanism limits performance.

Current operating systems, such as Windows, AIX and Linux, provide facilities for fast, scalable IO based on the use of asynchronous notifications of IO operations taking place in the operating system layers. For example, Windows and AIX have IO Completion Ports, while Linux has the sys_epoll facility. The Async IO package aims to make these fast and scalable IO facilities available to Java applications through a package that provides IO capabilities linked to an asynchronous style of programming.

The current version of the Async IO package, com.ibm.io.async, is designed as an extension to the Java 2 Standard Edition 1.4, which can in principle be provided on any hardware and software platform. The platforms currently supported by the package include Windows, AIX, Linux, and Solaris.

Elements of the Async IO Package

The major elements of the Async IO package are the classes AsyncFileChannel, AsyncSocketChannel, and AsyncServerSocketChannel. The channels represent asynchronous versions of files, sockets, and server sockets. These fundamental classes are designed to be similar in naming and in operation to the channel classes of the New IO package. Good news for Java programmers familiar with the New IO package.

AsyncFileChannels and AsyncSocketChannels provide asynchronous read and write methods against the underlying file or socket. An asynchronous operation is a request to the system to perform the operation, where the method returns immediately to the calling application regardless of whether the operation has taken place or not. Instead of providing a return value that gives information about the operation, such as the number of bytes read/written, asynchronous read and write operations return objects that implement the IAsyncFuture interface.

The IAsyncFuture interface is another important component of the Async IO package. First an IAsyncFuture represents the state of the asynchronous operation – most important, whether

the operation has completed or not. Second, the IAsyncFuture provides methods that return the result of the operation once it has completed. An IAsyncFuture can throw exceptions as well as the normal outcome of the operation, if something goes wrong during the operation.

The application uses one of three methods to find out whether a particular operation has completed:

- ***Polling:*** Calls the isCompleted() method of the IAsyncFuture, which returns true once the operation is complete
- ***Blocking:*** Uses the waitForCompletion() method of the IAsyncFuture, which can be used either to wait for a specified period or to wait indefinitely for the operation to complete
- ***Callback:*** Uses the addCompletionListener() method of the IAsyncFuture, so the application can register a method that's called back by the system when the operation completes

Which method an application uses to find out about the completion of an operation is driven by the overall design of the application, although the most truly "asynchronous" designs would tend to use the callback method. If an application uses the callback method, the thread that requests the IO operation carries on to do more work, while the callback is normally handled on a separate thread.

Data Formats Supported by Asynchronous Read and Write Operations

The read and write operations supplied by the Async IO package use the ByteBuffer class to hold the data. This class is the same as the one used in the New IO package. One difference between the Async IO package and the New IO package is that the ByteBuffers used for the Async IO package must be Direct ByteBuffers. Direct ByteBuffers have the memory for their content allocated in native memory outside the Java Heap. This provides better performance for IO operations since the operating system code can access the data in the buffer memory directly, without the need for copying.

ByteBuffers can be viewed as buffers supporting other primitive types, such as Int, Float, or Char, using methods such as bytebuffer.asIntBuffer(). ByteBuffers also have a series of methods that support the reading and writing of primitive types at arbitrary locations in the ByteBuffer using methods like bytebuffer.putLong (index, aLong).

Simple Examples of Async IO Read and Write Operations

Listing 1 shows the use of an AsyncSocketChannel as a client socket that involves connecting the socket to a remote server and then performing a read operation. In this example, the blocking style is used to wait for asynchronous operations to complete.

Listing 2 is a program fragment that shows the use of a callback to receive the notification of the completion of an asynchronous operation. This fragment shows just some of the methods of a class that is handling socket IO. It's assumed that an AsyncSocketChannel has already been opened and connected, that a direct ByteBuffer is available, and that an object named "state" tracks the state of the IO.

When the IO operation is requested (channel.read(...)) an IAsyncFuture is returned. The next step is to give

Tim Ellison is a senior software engineer and strategic planner in the emerging technologies team at IBM Hursley Java Technologies group. He has contributed to the implementation of Smalltalk, IBM VisualAge Micro Edition, Eclipse, and the Java SDK over a period of 20 years. His interests are in new ways to apply object technology to difficult problems.

tim_ellison@uk.ibm.com

the IAsyncFuture a callback method by calling the addCompletionListener (...) method. The callback method gets called when the operation completes. The callback method is the futureCompleted(...) method that forms part of a class that implements the ICompletionListener interface.

In this example, the class with the callback is the same as the class that makes the read request (so "this" is used as the first parameter in the addCompletionListener method). The signature of the futureCompleted (…) method is fixed: its parameters are an IAsyncFuture object that represents the operation and, second, an object that holds the application state, which is associated with the IAsyncFuture through the addCompletionListener(…) method where it forms the second parameter (in this example, we use the object called "state").

The futureCompleted(…) method is called when the operation completes. It is possible that the operation is complete before the completion listener is added to the future. If this happens, the futureCompleted(…) method is called directly from the addCompletionListener(…) method, without any delay.

The futureCompleted(…) method receives the future object relating to the completed operation, plus the application state object.

Beyond the Basics: Multi Read/Write Operations and Timeouts

The previous sections described the basic functions available as part of the Java Async IO package. The package also supplies more advanced interfaces for asynchronous IO. The first advanced interface supplies the capability to perform read and write operations using multiple buffers for the data. The second advanced interface provides a time-out on the asynchronous IO operation.

Both the multi read/write operations and the time-out facility are provided by the AsyncSocketChannelHelper and AsyncFileChannelHelper classes. This is done to keep the interface to the AsyncFileChannel and AsyncSocketChannel classes as straightforward as possible.

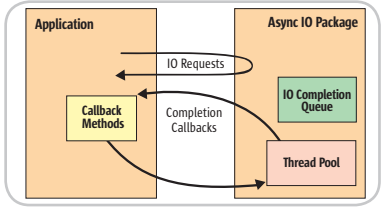


Figure 1 Async IO Thread Pool

Create an AsyncSocketChannelHelper object by wrapping an existing AsyncSocketChannel. An AsyncFileChannelHelper is created by wrapping an existing AsyncFileChannel object. All operations on the channel helper object apply to the underlying asynchronous channel.

The multi read/write operations take ByteBuffer arrays as input and return IAsyncMultiFuture objects. IAsyncMultiFuture objects differ from IAsyncFuture objects only in that they have a getBuffers() method that returns the ByteBuffer arrays involved in the operation in place of the getBuffer() method, which relates to the single buffer read/write operations. The multi read/write operations are useful for applications that need to send or receive data that's best handled by multiple buffers, perhaps where different elements of the data are handled by different application components (see Listing 3).

The time-out operations provided by the AsyncSocketChannelHelper and AsyncFileChannelHelper classes are versions of the basic read and write operations that have a time-out period applied to them. The basic read and write operations of asynchronous channels can in principle take forever to complete. This is particularly a problem for an application that uses the callback technique to get notified that the operation is complete, since the callback might never get called if the operation does not complete. The use of the time-out versions of the operations guarantees that the IAsyncFuture will complete when the time-out expires, even if the underlying read/write operation does not complete. If the time-out expires, the IAsyncFuture completes with an AsyncTimeoutException. In addition, the underlying operation is cancelled (equivalent to invoking the IAsyncFuture cancel(future) method).

Note that using the time-out versions of read and write are different from using the IAsyncFuture waitForCompletion(timeout) method (see Listing 4). waitForCompletion provides a time-out for the wait on the completion of the IAsyncFuture. If this time-out expires, control is returned to the application, but the IAsyncFuture is not completed and the underlying read/write operation is still underway. By contrast, if the time-out expires on the AsyncChannelHelper read/write methods, the IAsyncFuture is completed (with an AsyncTimeoutException) and the underlying operation is cancelled.

An important point about operations that time out is that the state of the channel is left indeterminate. Once an operation is cancelled, it's unlikely that the channel can be used again and the safe option is for the application to close the channel.

Asynchronous IO Thread Management

If you write an application program that uses the callback method to get notifications that asynchronous IO operations have completed, you need to understand which Java threads are used to run the callbacks. The threads used to run the callbacks will run application code. If your application code needs the threads to have any special characteristics, such as specific context information or security settings, this could cause problems for your application code unless your application carefully controls the actual threads that are used to run the callbacks.

The threading design of the Async IO package is outlined in Figure 1. Applications make requests to the package for Async IO operations. The requests are passed to the operating system's IO functions. When the operations complete, notifications of their completion are passed back to the Async IO package and are initially held in an IO Completion Queue. The Async IO package has a set of one or more Java threads that it uses to process the notifications in the IO Completion Queue. Notifications are taken from the Completion Queue, and the IAsyncFuture related to the operation is marked as completed. If a Callback Listener has been registered on the IAsyncFuture, the Callback Listener method is called. Once the CallBack Listener method finishes, the thread returns to the Async IO package and is used to process other notifications from the Completion Queue.

By default, the Async IO package uses its own Result Thread Manager to manage the threads that handle the callbacks. It allocates a number of threads, typically equal to the number of processors on the system. These threads are vanilla Java threads with no special characteristics. However, the application can control the threads in one of two ways.

The application can override the default Result Thread Manager by calling the setResultThreadManager(IResultThreadManager) method of the Abstract- AsyncChannel class. The application must supply its own manager class that implements the IResultThreadManager interface, which defines the full life cycle for threads used by the Async IO package. The IResultThreadManager interface provides control over the policies applied to the result threads, including the timing of creation and destruction, the minimum and maximum numbers of threads, plus the technique used for creation and destruction of the threads.

Number of Clients	Sync Server	New IO Server	Async Server
Each client performing 50 read/write cycles.	overall time/data transfer time	overall time/data transfer time	overall time/data transfer time
50	357/186	487/180	332/180
200	326/178	200/179	211/178
400	193/181	201/178	190/178
1000	200/180	433/197	222/186
3000	230/180	565/184	217/201
1,000 active + 3,000 static	232/187	688/196	230/181
1,000 active + 6,000 static	Failed out of memory	781/269	275/195
All times are average microseconds per cycle.			

Table 1 Performance comparison of IO packages

Alternatively, the application can use the default IResultThreadManager implementation provided by the Async IO package, but control the nature of the threads used to handle results and callbacks. This is done by supplying the default IResultThreadManager implementation with an application-defined IThreadPool object, by calling the set-ThreadPool(IThreadPool) method on the IResultThreadManager. This allows the application to control the nature of the threads used in the Result Thread Manager. For example, application data can be attached to the thread or specific security settings applied to the thread, or the threads used in the IResultThreadManager can be cached by the IThreadPool.

Performance

Performance is one of the important reasons for using the Async IO package. How does its performance stack up against the original synchronous Java IO and also against the New IO package?

Performance is a complex issue, but a simple test provides some guidance. The test uses Socket IO with multiple clients communicating with a single server. Each client performs repeated operations, writing 256 bytes to the server and reading a 2,048 byte response from the server. For the test, the clients are always the same code, but three variations of the server code are used:

- Synchronous Server, using the original Java IO classes
- New IO Server, using the New IO classes
- Asynchronous IO Server, using the Async IO package

The server code is as similar as possible, but the differences implied by the different programming models of the IO packages are built into the code. Most notably, the threading design of the Synchronous Server is one-thread-per-client, while the number of threads used for the New IO and Asynchronous Server is determined by the number of processors on the server system and is much smaller than the number of clients. An important feature of the server code is that the caches of the common objects used by the server code are used as much as possible – notably the ByteBuffers (in all three cases) and the threads (in the case of the Sync Server). This is done to reduce the startup times as much as possible for each socket; this reflects a common practice for typical server designs.

We ran the tests with a Windows 2000 single processor server system and a Windows Server 2003 four-way system running the clients, connected via a 100Mb Ethernet network, with varying numbers of client sockets each performing a connect followed by 50 read/write cycles with the server. The results are shown in Table 1, which provides the data for the average time in microseconds to complete each read/write cycle, quoted with and without the startup time included. The startup time is the time taken for the client socket to connect to the server before any data is transmitted.

(If you're surprised that the four-way server system is used to drive the client side for this test, it's used to ensure that the very large number of clients can be created successfully.)

The last two cases involve running with a number of inactive client sockets, which are connected to the server but are not transmitting any data during the test. This is more typical of a real Web server. These inactive sockets are a load for the server to handle alongside the active sockets.

This shows the Async IO, New IO, and Sync servers are all similar in terms of average times in lightly loaded situations. The failure of the Sync server to handle the case of 7,000 total clients shows its limitations in terms of scalability. The figures for the New IO server show that the performance suffers as the number of clients rise. In particular the New IO server shows a marked rise in the overhead for starting up new connections as the number of connections rises. The Async IO server manages to achieve reasonably stable performance right through the range tested, both for startup time and for the read/write cycle time.

These simple tests show that the Async IO package is able to deliver on its promise of performance and scalability and can form part of the solution for server applications intended to handle many thousands of clients.

Pitfalls to Avoid

As with the use of any API, there are some aspects of the Async IO API that you need to think about to avoid problems.

You need to be careful with the use of the ByteBuffers that are used in the read and write methods of asynchronous channels. Because the IO operations occur asynchronously, there is the potential for the Async IO package to use the ByteBuffers at the same time as the application code. The rule to follow in order to avoid trouble is that the application code should not access the ByteBuffers from the time that an asynchronous read or write operation is requested until the point that the Async IO package signals that the operation is complete. Any attempt by the application to access the ByteBuffers before the operation is complete could cause unpredictable results.

Asynchronous channels provide facilities for the cancellation of asynchronous IO operations. These include the explicit cancel() method available on the futures returned by operations on asynchronous channels, and also the implicit cancellation that takes place as part of the time-out of an IO operation on an

Listing 1: Basic Use of AsyncSocketChannel

```
        /* Create a new AsyncSocketChannel and connect it to a
 * given hostname and portnumber.
 *_____________________________________________________________
 */

try {
 AsyncSocketChannel aChannel = AsyncSocketChannel.open();
 InetSocketAddress theAddress = new InetSocketAddress( hostname, portnumber );
 IAsyncFuture future = aChannel.connect( theAddress );
 /* The connect operation is asynchronous
  * wait for completion on the future returned by the connect
  * method
  */
 future.waitForCompletion();
} catch ( Exception e ) {
 // Deal with exception while opening and connecting the Channel
} // end try block

// Perform a read operation

// First set up the Byte Buffer
ByteBuffer readbuf = ByteBuffer.allocateDirect( BUFFERSIZE );

try {
 // Request the read operation
 IAsyncFuture future = channel.read( readbuf );

 // wait for completion of read, up to 15 seconds
 long bytesRead = future.getByteCount( 15000 );
} catch ( AsyncTimeoutException te ) {
 // the wait for the read to complete timed out
} catch ( IOException ie ) {
 // an IOException happened when performing the read
} catch ( ClosedChannelException che ) {
 // the AsyncSocketChannel was closed before the read
} catch ( InterruptedException inte ) {
 // the thread was interrupted while waiting
} // end try block
```

AsyncSocketChannelHelper or AsyncFileChannelHelper. If an operation is cancelled, the underlying channel (file or socket) is left in an indeterminate state. Because of this, your application should not attempt to perform any more operations on the channel once cancellation has occurred. The best thing to do is to close the channel as soon as possible.

The performance of read and write operations using Async IO is designed to be as close as possible to the performance of equivalent synchronous IO operations. However, there is some extra overhead involved in running an asynchronous operation compared with a synchronous operation, associated with setting up and executing the asynchronous notifications. The implication of this is that asynchronous reads and writes involving very small packets of data (i.e., a few bytes only) are going to have a significantly higher overhead than synchronous equivalents. You should take this into account when designing your application to use Async IO.

Summary

The Java Async IO package provides valuable facilities for fast, scalable Socket and File IO, which are an alternative to the use of java.io and java.nio facilities in client-side and server-side applications. The package also assists the program design by providing an event-driven interface for IO operations that is simple to use.

Resources

- *The C10K Problem Page contains a comprehensive discussion of the need for fast, scalable IO for servers and the facilities available to provide this on various systems:* www.kegel.com/c10k.html
- *Pattern-Oriented Software Architecture:* www.cs.wustl.edu/~schmidt/POSA/
- *New IO APIs have a full description of the standard Java New IO package:* http://java.sun.com/j2se/1.4.2/docs/guide/nio/index.html
- *Download the Async IO package to try out in your applications:* www.alphaworks.ibm.com/tech/aio4j

Listing 2: Simple use of Callback with a future

```
public void performRead() {
 try {
 // read
  buffer.position( 0 );
  buffer.limit( buffer.capacity() );
  state = READ_PENDING;
  IAsyncFuture future = channel.read( buffer );
  // Set up the callback
  future.addCompletionListener( this, state );
 } catch (Exception e) {
  System.out.println("Exception occurred on read request");
  // Handle exception...
 } // end try
} // end method performRead

// Callback method dealing with completed operations
public void futureCompleted( IAbstractAsyncFuture theFuture,
Object userState ) {
 if ( userState == READ_PENDING ) {
  try {
   long bytesread = aFuture.getByteCount();
   //.... do processing ....
  } catch (Exception e) {
   //.... handle exceptions ...
  }
 } // end if

 if ( userState == WRITE_PENDING ) {
  try {
   long byteswritten = aFuture.getByteCount();
   //.... do processing ....
  } catch (Exception e) {
   //.... handle exceptions ...
  }
 } // end if
} // end method futureCompleted
```

Listing 3: Use of multi read operation

```
// Async Socket Channel: assume open and connected in earlier
code
AsyncSocketChannel s;

ByteBuffer[] bufs = new ByteBuffer[ MAXBUFS ];
// Create the AsyncSocketChannelHelper
AsyncSocketChannelHelper helper = new AsyncSocketChannelHelper( s
);

//Allocate the buffer array
for ( int i = 0 ; i < MAXBUFS ; i++ ) {
 bufs[i] = ByteBuffer.allocateDirect( BUFFERSIZE );
} // end for

try {
// read
 IAsyncMultiFuture future = helper.read( bufs );
 // Use blocking to wait for completion...
 long bytesRead = future.getByteCount( );
} catch (Exception e) {
 System.out.println("Exception occurred on read request");
 // Handle exception...
} // end try
```

Listing 4: Use of read operation with time-out

```
// Perform a read operation with a timeout
AsyncSocketChannel schannel;
static final int TIMEOUT = 15000;   // 15 second timeout

AsyncSocketChannelHelper helper = new AsyncSocketChannelHelper(
schannel );

// Set up the Byte Buffer
ByteBuffer readbuf = ByteBuffer.allocateDirect( BUFFERSIZE );

try {
 // Request the read operation
 IAsyncFuture future = helper.read( readbuf, TIMEOUT );

 // wait for completion of read, or for the timeout to happen
 long bytesRead = future.getByteCount( );
} catch ( AsyncTimeoutException ) {
 // deal with the timeout of the operation
} catch ( Exception e ) {
 // deal with the exception...
} // end try block
```

Java Certification

An introduction

by Naveen Gabrani

Have you looked at the certificate that your neighbor has so proudly displayed in his or her office? Have you ever wondered if getting certified in Java is worth the time and effort? This article argues the case for certification and provides information on how to prepare for the SCJP exam.

In today's tough job market, a certification from an established vendor like Sun can act as one more selling point in your résumé. All else being equal, most employers will prefer a certified candidate. Certification alone does not guarantee a job. But it acts as a reassurance to a prospective employer that the candidate at least understands the basics of Java. The certification is especially useful if you are a new graduate or if you're looking for your first job in Java.

Perhaps even more useful than getting the certificate is the learning involved in preparing for the exam. The certification enforces discipline. Most people are too busy with their jobs to have time to stay up-to-date on the latest technologies. Preparing for certification helps bring focus and discipline, and lets you update your skills on a specific technology. The certification exams from Sun are industry standard and well recognized in the industry.

Sun offers seven Java-related certification exams:

- Sun Certified Java Programmer (SCJP)
- Sun Certified Java Developer (SCJD)
- Sun Certified Web Component Developer (SCWCD)
- Sun Certified Business Component Developer (SCBCD)
- Sun Certified Mobile Application Developer (SCMAD)
- Sun Certified Enterprise Architect (SCEA)
- Sun Certified Developer for Java Web Services (SCDJWS)

Sun also has another new exam – Web Services Certification (SCDJWS). This is currently in beta and is likely to be launched in the near future.

A quick overview of each of these exams is provided in the following.

Naveen Gabrani is a project manager at Computer Sciences Corporation (CSC), India. He also runs a Java certification site www.javaprepare.com.

ngabrani@csc.com

***Sun Certified Java Programmer* (SCJP)** is the prerequisite for most of the other exams. It tests your knowledge of the core Java language. There are three versions of the SCJP exam – Java 1.2, Java 1.4, and an upgrade from Java 1.2 to Java 1.4. Unless you are using AWTs extensively, you should stick to the SCJP exam for Java 1.4.

Main Contents	Java Fundamentals, Threads, Collections
Exam Number	CX-310-035
URL	http://suned.sun.com/US/catalog/courses/CX-310-035.html
Prerequisites	Nil
Exam Fee	$150
Exam Type	Multiple Choice
Number of questions	61
Minimum to Pass	32
Time Limit	2 hours

***Sun Certified Java Developer* (SCJD)** is a more advanced exam that covers J2SE (Standard Edition). The exam consists of two steps – a take-home programming assignment and an essay exam on the programming assignment. The essay exam is conducted through the Prometric center. The exam requires more time and money commitment than most of the other Java Certification exams.

Main Contents	Object-Oriented Design, GUI, and Locking
Exam Number	CX-310-025A (programming assignment) and CX-310-027 (online essay exam)
URL	http://suned.sun.com/US/certification/java/java_devj2se.html
Prerequisites	SCJP
Exam Fee	$250 for programming assignment and $150 for the essay exam
Exam Type	Programming assignment and essay exam
Number of questions	4 questions in the essay exam
Minimum to Pass	320 out of 400.
Time Limit	2 hours for essay exam. No time limit for programming assignment.

***Sun Certified Web Component Developer* (SCWCD)** exam mainly covers servlets and JSP.

Main Contents	Servlets, JSP, and Design Pattern
Exam Number	CX-310-081
URL	http://suned.sun.com/US/catalog/courses/CX-310-081.html
Prerequisites	SCJP
Exam Fee	$150
Exam Type	Multiple Choice
Number of questions	69
Minimum to Pass	43
Time Limit	135 minutes

***The Sun Certified Business Component Developer* (SCBCD)** exam was introduced a year back. It focuses on various facets of EJBs. The exam consists of multiple-choice questions on designing, developing, testing, deploying, and integrating Enterprise JavaBeans.

Main Contents	EJBs
Exam Number	CX-310-090
URL	http://suned.sun.com/US/certification/java/java_busj23e.html
Prerequisites	SCJP
Exam Fee	$150
Exam Type	Multiple Choice
Number of questions	70
Minimum to Pass	45
Time Limit	120 minutes

***Sun Certified Mobile Application Developer* (SCMAD)** exam focuses on the micro edition (J2ME) – Sun's platform for mobile Java technologies. This exam was introduced recently.

Main Contents	Wireless Messaging API and Mobile Media APIs
Exam Number	CX-310-110
URL	http://suned.sun.com/US/catalog/courses/CX-310-110.html
Prerequisites	SCJP
Exam Fee	$150
Exam Type	Multiple Choice
Number of questions	68
Minimum to Pass	38
Time Limit	150 minutes

***Sun Certified Enterprise Architect* (SCEA)** exam evaluates architect-level skills on the J2EE platform. The exam has three parts – a multiple choice–based exam (Part 1), a programming assignment (Part 2), and an essay type (Part 3) that has follow-up questions on the programming assignment. You must clear all three to be certified. It's arguably the most complex Java certification exam offered by Sun.

Main Contents	J2EE
Exam Number	Multiple Choice (CX-310-051), Assignment (CX-310-300A), Essay exam (CX-310-061)
URL	http://suned.sun.com/US/certification/java/java_archj2ee.html
Prerequisites	None
Exam Fee	$150 for part 1, $250 for part 2, $150 for part 3.
Exam Type	Multiple Choice, programming assignment and essay type
Number of questions	48 (part 1), 4 (part 3)
Minimum to Pass	31 (part 1), 70% for the essay and assignment.
Time Limit	75 minutes (for part 1), 90 minutes (for part 3)

***Sun Certified Developer for Java Web Services* (SCDJWS)** has just been launched by Sun. It covers Java technology components related to Web services.

Main Contents	XML Web Service Standards, SOAP, WSDL, UDDI, JAX-RPC, JAXR
Exam Number	CX-310-220
URL	http://suned.sun.com/US/catalog/courses/CX-310-220.html
Prerequisites	SCJP
Exam Fee	$250 for programming assignment and $150 for the essay exam.
Exam Type	Multiple Choice
Number of questions	69
Minimum to Pass	47
Time Limit	150 minutes

Scheduling the SCJP Exam

All the online exams are conducted at the Sylvan Prometric Centers throughout the world. To schedule a test:

1. Visit http://suned.sun.com/US/catalog/courses/CX-310-035.html. Add the product to the shopping cart and purchase it.
2. Visit www.prometric.com. Locate a Prometric site that suits you. After this you can schedule an exam through the site.

Course Contents

The exam objectives define the course contents for the exam. The topics include declarations and access control, flow control, assertions and exception handling, garbage collection, language fundamentals, operators and assignments, overloading, overriding and object orientation, threads, fundamental classes, and collections.

The exam covers some of the facets of Java that are sometimes unknown even to a seasoned Java programmer. So some exam-specific preparation is required even for experienced Java programmers.

Sample Questions

Question 1 (Declaration and access control)

Which of the following are valid constructors within a class Test? Select the two correct answers.

A. test() { }
B. Test() { }
C. void Test() { }
D. private final Test() { }
E. abstract Test() { }
F. Test(Test t) { }
G. Test(void) { }

Answer: B and F. A constructor must have the same name as the class. The name of the class is Test. Since Java is case sensitive, test is different than Test, hence option A is not valid. The constructor must not return any value, so option C is incorrect. A constructor cannot be declared abstract or final. Hence D and E are incorrect answers. Since taking a void argument is illegal in Java, G is also illegal.

Question 2 (Flow control)

What is printed when the following gets compiled and run? Select the two correct answers.

```
1. public class example {
2. public static void main(String args[]) {
3.      int x = 0;
4.      if(x > 0) x = 1;
5.
6.      switch(x) {
7.           case 1: System.out.println(1);
8.           case 0: System.out.println(0);
9.           case 2: System.out.println(2);
10.        break;
11.   case 3: System.out.println(3);
12.        default: System.out.println(4);
13.        break;
14. }
15. }
16. }
```

A. 0
B. 1
C. 2
D. 3
E. 4

Answer: A and C. The main method initialized *x* to 0. The if condition at line 4 returns false, so *x* remains zero at the beginning of the switch statement. The value of *x* matches with case 0. Hence 0 gets printed. Since there is no break statement after this, 2 also get printed. The break statement after case 2 leads to the control coming out of the switch statement. Hence 0 and 2 are the only numbers that get printed.

Question 3 (Assertions and exception handling)

What happens when the following code is compiled and run? Select the one correct answer.

```
1. public class example {
2. public static void main(String args[]) {
3. for(int i = 1; i < 4; i++)
4.    for(int j = 1; j < 4; j++)
5.      if(i < j)
6.        assert i!=j : i;
7. }
8. }
```

Is your SOA really agile?

Building, testing and maintaining SOA solutions requires a flexible and collaborative approach to diagnostics. Finger pointing and confusion occur when all parties do not have a complete and common understanding of a problem. Mindreef® SOAPscope® 4.0 connects developers, testers, support personnel, and operations by combining the testing and diagnostic tools for each discipline and making it easy to share complete problem data. SOAPscope is the only end-to-end diagnostic system for Web services.

Developers • Testers • Operations • Support

NEW! SOAPscope 4.0

Test - Agile testing approach for both developers and testers allows what-if scenarios without coding or scripting. Includes the industry's most complete interoperability suite and the flexibility to really test the edge cases.

Capture - Easily capture and assemble complete problem data into SOAPscope's workspace. Capture SOAP messages off the wire, test cases, or re-produced problems. Capture a snapshot in time of a WSDL including all related resources.

Share - Exported workspaces support sharing among team members preserving complete problem data including messages, WSDLs, and notes. Sharing workspaces is a great way to support web service users.

Solve - Diagnostic suite of analysis, comparison and visibility tools simplifies the understanding of complex XML, yet are powerful enough for the industry's most protocol-savvy experts.

"SOAPscope has been invaluable in testing and diagnosing our enterprise scale Web Services. The new Workspace feature in SOAPscope 4.0 is an excellent way to support customers and collaborate as a team. A must have for every WS-Developer"

Simon Fell
Senior Member of Technical Staff at salesforce.com and author of PocketSOAP

Try SOAPscope **FREE** at www.Mindreef.com

Mindreef, Inc.
www.Mindreef.com

A. The class compiles and runs, but does not print anything.
B. The number 1 gets printed first with AssertionError.
C. The number 2 gets printed first with AssertionError.
D. The number 3 gets printed first with AssertionError.
E. The program generates a compilation error.

Answer: A. The if condition returns true, if *i* is less than *j*. The assert statement performs a check whether *i* is equal to *j*. Since *i* will never be equal to *j*, the assert statement will always return true. Hence the AssertionError does not get generated. So B, C, and D are incorrect.

Question 4 (Garbage collection)

At what stage in the following method does the object initially referenced by *s* become available for garbage collection? Select the one correct answer.

```
1. void method X()  {
2.      String r = new String("abc");
3.      String s = new String("abc");
4.      r = r+1;
5.      r = null;
6.      s = s + r;
7. }
```

A. Before statement labeled 4
B. Before statement labeled 5
C. Before statement labeled 6
D. Before statement labeled 7
E. Never

Answer: D. After statement 3, *r* and *s* are pointing to two different memory locations both containing "abc". After statement 4, the memory location to which *r* was pointing is available for garbage collection. After statement 6, *s* now points to a different location. The memory block that *s* was pointing to that contained "abc" is now available for garbage collection.

Question 5 (Language fundamentals)

Which of these are legal identifiers? Select the three correct answers.
A. number_1
B. number_a
C. $1234
D. -volatile

Answer: A, B, and C. In Java, each character of an identifier can be either a digit, letter, underscore, or a currency symbol. The first character cannot be a digit. Since – is not a valid character in an identifier name, D is incorrect.

Question 6 (Language fundamentals)

What happens when the following program is compiled and executed with the command - java test? Select the one correct answer.

```
1. class test {
2. public static void main(String args[]) {
3. if(args.length > 0)
4.      System.out.println(args.length);
5. }
6. }
```

A. The program compiles and runs but does not print anything.
B. The program compiles and runs and prints 0.
C. The program compiles and runs and prints 1.
D. The program compiles and runs and prints 2.
E. The program does not compile.

Answer: A. The above program is a legal Java file. No arguments are passed to the program, as the command line just says java test. test here is the name of the class. The args array does not contain any elements. Since args.length is zero, the program does not print anything.

Question 7

What happens when the following program is compiled and run? Select the one correct answer.

```
1. public class example {
2.     int i[] = {0};
3.     public static void main(String args[])
{
4.        int i[] = {1};
5.        change_i(i);
6.        System.out.println(i[0]);
7.     }
8.     public static void change_i(int i[]) {
9.        i[0] = 2;
10.      i[0] *= 2;
11.    }
12. }
```

A. The program does not compile.
B. The program prints 0.
C. The program prints 1.
D. The program prints 2.
E. The program prints 4.

Answer: E. The example is a legal Java class so A is incorrect. When array is passed as an argument, Java treats it as a call by reference. When the method change_i updates i[0], the first element of the array is directly getting modified. This element is set to 4 in change_i. This change in the value of i[0] is available in the main method also. Hence 4 gets printed.

Question 8 (Operators and assignments)

In the following class definition, which is the first line (if any) that causes a compilation error? Select the one correct answer.

```
1. public class test {
2.  public static void main(String args[]) {
3.    char c;
4.    int i;
5.    c = 'A';
6.    i = c;
7.    c = i + 1;
8.    c++;
9.  }
10. }
```

A. The line labeled 5
B. The line labeled 6
C. The line labeled 7
D. The line labeled 8
E. All the lines are correct and the program compiles

Answer: C. "A" is a character. So assigning "A" to c in statement 5 is perfectly valid. Assigning a character to integer is also valid and does not require a cast. Thus the statement labeled 6 does not generate a compilation error. It's not possible to assign an integer to a character without a cast. Hence the statement 7 generates a compilation error.

Question 9 (Objects)

Which of the following are true? Select the three correct answers.
A. A static method may be invoked before even a single instance of the class is constructed.
B. A static method cannot access nonstatic methods of the class.
C. An abstract modifier can appear before a class or a method but not before a variable.
D. The final modifier can appear before a class or a variable but not before a method.
E. A synchronized modifier may appear before a method or a variable but not before a class.

Answer: A, B, and C. A and B are correct statements about the behavior of static methods. The abstract modifier may appear before a method or a class, but not before a variable, so C is also correct. A final modifier may appear before a method, a variable or before a class. Hence D is incorrect. A synchronized modifier cannot come before a variable so E is also incorrect.

Question 10 (Collections)

Which of these are interfaces in the collection framework? Select the two correct answers.

A. HashMap
B. ArrayList
C. Collection
D. SortedMap
E. TreeMap

Answer: C and D. Collection and SortedMap are examples of interfaces in the collection framework. HashMap. ArrayMap and TreeMap are examples of general purpose implementations for some of the collection interfaces.

Question 11 (Fundamental classes)

What gets written on the screen when the following program is compiled and run? Select the one correct answer.

```
1. public class test {
2.     public static void main(String args[]) {
3.     int i;
4.     float  f = 2.3f;
5.     double d = 2.7;
6.     i = ((int)Math.ceil(f)) * ((int)Math.round(d));
7.
8.     System.out.println(i);
9.     }
10. }
```

A. 4
B. 5
C. 6
D. 6.1
E. 9

Answer: E. The method ceil returns the smallest double value equal to a mathematical integer that is not less than the argument. Hence ceil(2.3f) returns 3. The round method round returns the integer closest to the argument. Hence round(2.7) returns 3. Multiplying the two numbers returns 9.

Resources for Preparing for SCJP

There are a large number of mock exams available on the Web. The links to the best of these is included below.

- Sun's sample questions for the programmer exam: http://suned.sun.com/US/certification/
- JavaRanch has extremely popular discussion forum on Java certifications and other topics: www.javaranch.com/certfaq.jsp
- Javaprepare has two mock exams, a tutorial and FAQ on Java certification: www.javaprepare.com
- Dan Chisholm's Web site has high-quality mock exams for the SCJP 1.4 exam. It has more than 350 questions. It also has mock exam questions by topic: www.danchisholm.net
- Javacertificate.com has a large pool of questions and some are organized by objectives: www.javacertificate.com

The three standard books on Java certification are:

- Mughal, K. A., Rasmussen, R. (2003). A *Programmer's Guide to Java Certification*. Addison Wesley Professional.
- Sierra, K., and Bates, B. (2002). *Sun Certified Programmer & Developer for Java 2 Study Guide*. McGraw-Hill Osborne Media.
- Roberts, S., and Heller, P., (2003). *Complete Java 2 Certification Study Guide, Fourth Edition*. Sybex Inc.

Any one of these will be sufficient for you to prepare for the exam.

Besides the numerous free mock exams available on the Web, there are a few commercial exam simulators also available. Some of the main ones are:

- Practice exam offered by Sun: http://suned.sun.com
- J@Whiz is a test simulator on the latest pattern of SCJP2, with 700 questions (12 Tests) at $59.95 or India Rs. 495: www.whizlabs.com
- JQPlus offers an exam simulator based on the new certification pattern: www.enthuware.com/jqplus
- JCertify provides an exam simulator that costs $69.95: http://enterprisedeveloper.com/jcertify

Other Java Certifications

Though Sun's Java certification is an industry standard, there is another certification exam on Java technology offered by Brainbench (www.brainbench.com/xml/bb/common/testcenter/taketest.xml?testId=118) that costs $49.95.

Retaking the Exam

In case you fail to clear the exam on the first attempt, you can take the exam again. You would need to buy another voucher from Sun. Sun also requires that you wait for two weeks before retaking the exam.

After Clearing the Exam

After clearing the exam, Sylvania Prometric will provide you with a score sheet highlighting your scores in various sections. You should receive a formal certificate from Sun. This may take 8–10 weeks. If you don't receive the certificate within this time frame, contact them. You may also check your certification results online at www.galton.com/~sun. This site has a database of all candidates who have cleared Sun's certifications.

Java Object Serialization for Performance

Streamline your applications

by John Cahill

Performance is a goal that all Java developers should put at the head of their design list for any project. To help you achieve this goal, there is a powerful mechanism that's used in just about every commercial Java product that most developers are not aware of. This mechanism is called serialization.

This article discusses Java object serialization and presents it as a possible performance-enhancing tool for Java systems. We'll discuss what serialization is and review a real-world case study where serialization was used to enhance the performance of an application. (The source code for this article can be downloaded from www.sys-con.com/java/sourcec.cfm.)

What Is Serialization?

Serialization has been around for a while. Simply put, it's a transmission and storage mechanism for Java objects. It was originally invented for use in RMI (Remote Method Invocation) so that Java objects could be reliably moved between servers. It's also used in many other Java systems, like JMS (Java Message Service), as a persistence mechanism for message queue objects.

These days the Java community largely ignores serialization. This is mainly due to long-term object storage issues and the fact that the serialized object is written out in native bytes. However, if you are aware of these issues, serialization can be an excellent solution to certain types of performance bottlenecks, if used wisely.

Serialization is a tricky process for the Java platform since an object can be a composite of other objects. For serialization to work correctly, all objects that are contained by the object you want to serialize must implement the serializable interface. The serializable interface has no methods that you're required to implement, as it's a marker interface used by the Java Virtual Machine to identify objects that can be serialized.

Development Scenario

A Case Where Serialization Helped to Enhance Performance

I started taking serialization seriously when I wrote a Swing application that featured some 2D API graphics. The application allowed users to view flat, wire-frame abstractions of vehicles and to pick which part of the vehicle they wanted information on. The application used animation, showing vehicle wire frame drawings, which were JPEG images. So when a mouse pointer was moved across the active pane, the application highlighted specific areas of the vehicle image.

To set up this feature I first had to map the vehicle wire-frame image "hot spots" and save the polygon points (*x*:*y* coordinates) to a file. There were a lot of polygons in each drawing, and a lot of *x*:*y* coordinates to contend with. When the program started, I found that reading in all of this static image setup data was a big performance hit. The images would take a few seconds to load, mainly because of reading in the image hot spot data. Since the image data never changed, serialization seemed like a good way to minimize this bottleneck. It turned out to be an excellent solution. Now the application uses a serialized collection of polygon objects for each type of image, and the image information loads immediately. If your application uses large amounts of static data, serialization might be the way to go.

How to Serialize Objects

The Basics, Part I: Serialization

After I realized that I had a performance problem, I needed to give my polygon collection the ability to serialize itself. While many Java objects are serializable, this is only the starting point for our serialization adventure.

First some bad news: the Java language gives developers a small set of tools to perform object serialization. Serialization is not inherently a part of the Java language as java.lang.Object contains no serialization methods. Now some good news: basically, object serialization is quite simple. There are only three steps to performing simple object serialization.

Step 1: Create an output file to hold the contents of the serialization stream. The name of the file doesn't matter since "the stream is the thing" (to paraphrase Shakespeare). It is, however, customary to name the file the same as the class name, with an extension of ".ser". This is the convention used in most of the Java platforms that use serialization.

John Cahill currently works as a core software engineer for Intuit Eclipse in Boulder. He has developed software in Java since 1995, and designed effective object-oriented solutions for a variety of businesses. He enjoys the Colorado lifestyle with his wife, Holly.

john_cahill_boulder@msn.com

Step 2: Create an ObjectOutputStream. This is a special type of OutputStream that performs the work of transforming the object into a stream of bytes that is specific to the Java serialization standard.

Step 3: Write the stream out and close the stream.

In real-word systems it would be inconvenient to have serialization as part of an application framework. You could do this, but we'll just use an abstract base class that supports serialization as in Listing 1.

The Basics, Part II: Deserialization

After I created the SerializableObject base class for my polygon collection, I needed some way for an application to read the serialized object from the file system. Whatever you serialize, you must be able to deserialize.

We will not give our SerializableObject class the ability to deserialize itself. If the user creates the object to deserialize another instance of the same object, what would be the point? It would be wasteful. The only way to make the method work would be to make the method static, which wouldn't work since it would be necessary to have access to the "this" reference, which static methods cannot use.

For the developer, deserialization can be broken down into the following programmatic tasks (see Listing 2). (Listings 2–3 can be downloaded from www.sys-con.com/java/sourcec.cfm.)

- Locate the serialized file of the object you are trying to deserialize.
- Get an InputStream to the file.
- Create an ObjectInputStream given the InputStream you have obtained in the previous step.
- Read in the object from ObjectInput.readObject(), performing the cast to the type of object and watching out for exceptions.

Using the Transient Keyword to Your Advantage

Let's say your object graph contains time/date sensitive info, or nonsecure items like database URLs or application user information. If this is the case, declare the fields in question to be transient. The Java serialization mechanism will exclude transient fields from the serialization process. When these fields are deserialized, they have a null value, if they are objects. The tricky bit here is to code your class in such a way that it is not critically dependent upon transient fields. I'd suggest that if your serialization candidate object contains lots of transient data, don't consider the object as a candidate for serialization.

Serializable vs Externalizable

Java provides a wonderful option to basic serialization. If you don't want your objects represented as a byte array, you should consider externalizable. The main difference between serializable and externalizable is that externalizable objects must provide instructions on how they are to be serialized and deserialized. This is a natural if you want absolute control over how your objects are represented in serialized form. This is also useful if you want to transform your objects to an abstract representation, like XML.

Speaking of XML, Java now gives you the choice to serialize your objects into XML rather than nonportable byte streams. See http://java.sun.com/j2se/1.4.2/docs/api/java/beans/XMLDecoder.html for information on this class, which can be used as a functional replacement for ObjectInputStream. The complimentary object is called XMLEncoder.

Serialization Pitfalls

All of this sounds great, but there are some practical downsides. The first pitfall is that serialization is a poor mechanism for the long-term storage of objects and their data. Why? It's quite simple. If your program relies on a serialized object, and the object changes (methods or fields deleted), you cannot deserialize that object anymore.

One obvious way around this is to never change that object. In practice, this is easier said than done, especially if another development team inherits the application. According to the serialization specification, there are both acceptable and unacceptable changes to a class. Simply put, you can add methods and fields to a class, but you cannot remove fields and methods and still expect serialization to work. Some methods to avoid this serialization pitfall might include:

- ***Serialize Java utility objects, like Properties, Hashtables, etc.*** Still, Sun Microsystems could change the implementations of these objects and then your code would not be able to run in a new version of the JDK.

- ***Version your objects with a unique numerical ID that can be read during serialization/deserialization.*** Sun does this with SDK objects that can be serialized and ships a tool called serialver that generates serialization IDs for classes. The ID is a hash of the class attributes (like class name, method count, etc.), which is a unique fingerprint for that class. Therefore, it's possible to compare the UID of a required object (referring to the required implementation) to ensure that it agrees with the serialized ID of the deserialized object. This approach is good to use if you plan to use externalizable object serialization and is not required of serializable objects.
- ***Late binding of serialized objects is the best practice.*** In other words, don't store the serialized object in a JAR file where people will forget about it. Have the application do the serialization work. That way the first time the app is run, the heavyweight data object is loaded, then serialized. On subsequent application runs, the application will look for serialized instances of the heavyweight data object. Also, consider using a class that implements the Factory pattern. That way the Factory can make an informed decision whether or not to use the serialized instance of the class. If you're not willing to take the "first time" performance hit, I suggest you consider generating the serialized objects at project build time. This way, the serialized objects can be generated and stored in a JAR at build time and will be valid for the lifetime of the compiled project codebase. This involves a little more work, though, because you have to build a "builder" for your serialized objects that can be called from Ant.

Setting Up a Framework for Serialization

Once I had gotten this far in my project, I took a step back to review my work. I had accomplished a very useful thing, but I was still dissatisfied. I had the polygon collection deciding if it should load the list of *x*:*y* coordinates from a file or use the serialized version of itself. This was a tad inefficient, to say the least. I decided to delegate this work to a singleton that controlled access to my polygon collection object.

Since late binding of serialized objects will keep you out of some types of serialization pitfalls, let's talk about how to control serialization in your application. This type of control is necessary because if you aren't careful, object serialization can turn out to be a nuisance.

Setting up a simple object factory is a good way to control access to your object and allow you to decide when it is appropriate to use the serialized version of the object in question. The other thing that makes this solution powerful is that you avoid the long-term storage issue of your object. If your application is performing serialization at runtime, the serialized object is never out of date. In Listing 3, a mechanism (like using serial UIDs) could be built in so that the object in question is only serialized where appropriate. When the object changes, the serialized object could be deserialized.

Some Serialization Trivia

Sun Microsystems has a serialization system used for JavaBeans that is part of the 1.4 SDK (see JSR 57 for more details). Note that this system is not part of the original serialization system. JSR 57 addresses long-term storage of JavaBeans and specifies how to serialize objects to abstract object representations (like XML), rather than byte streams. Also, the serialization system for Java was designed so that different byte encodings could be used, since these encodings on other platforms could differ.

Configuration Management Issues

After reading all this information, you have decided whether to actually store serialized objects in resource JARs (or on servers, where they can be accessed via URLs). If you do this, remember to play nice with your local configuration management specialist. For CM people and anyone else who is responsible for installing and migrating applications, serialized objects represent an odd problem: serialized objects are not any of the following:

- Source code
- Application properties
- Compiled code

I would like to offer this bit of advice: if you decide to include serialized objects as an application resource, they should be generated at build time. For this scenario I would recommend that you get involved with creating a custom Ant target that creates, serializes, and packages your serialized objects.

Again, the biggest pitfall here, and it is a significant one, is guarding against objects going out of date, or if the code base has been repackaged.

The other thing to be aware of is the practical impossibility of building on one platform and deploying on another. Since the serialized object stream uses the native platform byte representation, it would be unwise to build your code on Windows (ASCII) when you intend to use the serialized class on an AS/400 (EBCDIC). This is why most serialization strategies used in other Java platforms serialize objects only when they are created/received on the platform that the code is running on.

Summary

I hope I have shown that serialization is a powerful tool with a few manageable limitations. I believe that it can be used in a variety of development scenarios, where object creation at runtime represents a performance problem. If you understand the limitations of serialization, using this tool can be a powerful ally that will help to streamline your applications.

Acknowledgment

Thanks to Chris Merrion of Intuit Eclipse for proofreading assistance.

Resources

- *JSR-57:* www.jcp.org/en/jsr/detail?id=57
- *Java Object Serialization Specification:* http://java.sun.com/j2se/1.4.2/docs/guide/serialization/spec/serialTOC.html
- *Ant Programming – see the "Writing your own task" section:* http://ant.apache.org/manual/index.html

Listing 1

```
package com.cathail.serialization;

import java.io.File;
import java.io.ObjectOutputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.Serializable;
import java.io.NotSerializableException;
import java.io.FileNotFoundException;

public abstract class SerializableObject  implements Serializable
{
    private String pathToSerializeTo = null;

    /**
     *  a basic, no frills serialization method. Note that
     *  it is customary to use ".ser" as a file extension
     *  for the serialized object filename
     *
     */
    public void serialize()
    {
        ObjectOutputStream os = null;

        try
        {
            /* set the output path for the object, if one has
been provided...
                if one has not been provided, just go without
it...
             */
            String outPath = ((this.pathToSerializeTo == null) ?
                              this.getClass().getName() :
                                 this.pathToSerializeTo +
                                 File.separatorChar +
                                 this.getClass().getName());

            File serializedObjectFile = new File(outPath +
".ser");

            os = new ObjectOutputStream(new FileOutputStream(seri
alizedObjectFile));

            /*
               also, you can encrypt the bytes coming off
               of the stream for added security!
             */

            os.writeObject(this);
        }
        catch (NotSerializableException ex)
        {
            /* it is important to catch this exception, rather
             * than just catching IOException, because even
though
             * this base class implements Serializable, it is
possible
             * to construct an object graph that contains non-
Serializable
             * objects, which would cause us some problems...
             */
            ex.printStackTrace();
        }
        catch (FileNotFoundException ex)

 {
            ex.printStackTrace();
        }
catch (IOException ex)
        {
            ex.printStackTrace();
        }
        finally
        {
            try
            {
                if (os != null)
                {
                    os.close();
                }
            }
            catch (IOException ex)
            {
                ex.printStackTrace();
            }
        }
    }

    /** this method will set the output path so that
      *   when serialization takes place, the output will go to
      *   a specific directory.
      *
      * @param path - the path to the serialized output
      */
    public void setSerializationPath(String path)
    {
        this.pathToSerializeTo = path;
    }

}
```

Joe Winchester
Desktop Java Editor

Private Conversations in Public

One of the principles of any OO language such as Java is an object's ability to encapsulate its data and provide clients with a specific and well-defined API. This is done through the visibility keywords public, protected, and private. The use of these is one of the first things any Java programmer learns; the fairly well-understood key points being that a public accessor (be it a method or a field) is visible to any other object, protected only to subclasses and private to no one but the class itself.

A not-uncommon scenario, however, is that when you're using a class, you find a gem of a method you want to call that's been marked as private, or you want to access or modify a field that isn't visible to your class. I find this occurs quite a bit with frameworks where you might be extending or calling someone else's code and it works fairly well except for one little twist that you need to complete your task. There is a way (that I describe below) that allows you to access private methods and fields and it's a useful technique that, in these situations, enables you to code your way out of a blind alley.

The secret lies in the java.lang.reflect package that lets you execute Java code reflectively. Typically this is used for such tasks as looking up a field, method, or constructor by name and then accessing it for a given object. It's very useful if your program has to consume objects at runtime that it has no prior knowledge of at compile time (like introspection, for example). The three classes – java.lang.reflect.Method, Field, and Constructor – however, all inherit from the java.lang.AccesibleObject. This has the method setAccessible(boolean) whose method comment is:

```
   * Set the "accessible" flag for this
object to
   * the indicated boolean value. A value of
"true" indicates that
   * the reflected object should suppress
Java language access
   * checking when it is used. A value of
"false" indicates
   * that the reflected object should
enforce Java language access checks.
```

To best way to illustrate this is to take class A, which has a private method isFooBar();

```
public class A{
  private boolean fooBar;
  private boolean isFooBar(){
   return fooBar;
  }
}
```

Using reflection we can get the method and execute it as follows:

```
A object = new A();
Method fooBarMethod = A.class.getDeclaredMet
hod("isFooBar");
fooBarMethod.setAccessible(true);
try{
 System.out.println("FooBar=" + fooBar-
Method.invoke(object);
} catch (Exception exc){
}
```

The key is the setAccessible(true) statement. If this weren't present, a java.lang.IllegalAccessException would be thrown. With the accessible set to true the code to execute the method and get the result can be in any class in the JVM. In addition to calling private methods, we can also set field values; the following code sets it to false:

```
Field fooBarField = A.class.getDeclaredFiel
d("fooBar");
fooBarField.setAccessible(true);
try{
 fooBarField.set(object,Boolean.FALSE);
} catch (Exception exc){
}
```

We're using the method setAccessible(boolean) as it was designed to allow reflection to ignore any visibility constraints; however, the setAccessible(boolean) method does go through the security manager, which throws a SecurityException if this kind of private invocation is now allowed. The default JRE security manager does not prohibit the accessibility being overridden, and if you wanted to check your JVM to see whether or not this technique was going to work, the code would be:

```
try{
 System.getSecurityManager().
checkPermission(new ReflectPermission(""));
} catch (SecurityPermission exc){
 // VM not going to let you do funky reflec-
tion
}
```

I don't recommend using reflection and accessibility as a matter of normal programming, but when you're in a tight spot with a deadline breathing down your neck and you can see the API you need but it's just not there, it can be a lifesaver. You'll be linking to private field names and methods though so the next version of the framework or class library you're abusing might very well change and break your code. You won't pick up this breaking change at compile time because it's reflective and will only die at runtime.

It's very interesting that this month we have a great article on Java 2D and gaming, and last month we published a fantastic piece on Java3D using Mars rover images as examples. I hope this is a renaissance in both frameworks and we see more interesting uses in which Java apps can find their way onto the desktop.

Joe Winchester is a software developer working on WebSphere development tools for IBM in Hursley, UK.

joewinchester@sys-con.com

Ignorance isn't bliss.

Knowing about performance problems before they happen is.

It might seem like bliss to ignore application performance problems and deal with them only if and when they surface. But waiting until a problem arises in production only results in crises, stress, and wasted time. Experience real bliss by eliminating performance problems proactively with DiagnoSys.

From development through production, DiagnoSys targets potential problems by providing a wide breadth of information, collecting data not only from the application server but also the associated resources, such as the Web server, database, and network.

This breadth of data gives a more accurate picture of a developing problem. Other products only look at the app server and imply performance of the resources based on the performance of the connections, a method that often misses the real source of the problem.

DiagnoSys also provides in-depth, statistically correlated information so you see a more complete view. DiagnoSys gathers information down to the method on the app server as well as in-depth information for all resources down to the OS.

Even though you get depth of information, you won't be overwhelmed. DiagnoSys synthesizes the data and quickly identifies the root cause so you can correct the problem before it impacts you or your company.

diagnoSys

Register for an upcoming Web seminar. Find ways to improve your Web app projects while driving profitability. Guest speakers share their knowledge. Register now at: www.hwcs.com/jdj04.asp

SILVER ANNIVERSARY 1979 - 2004

H&W

www.hwcs.com | 1.800.338.6692

Bridging the Enterprise Computing Gap For 25 Years

JAVA GAMING

Understanding the basic concepts Part 1

by Chet Haase
and Dmitri Trembovetski

At JavaOne 2004 we gave a presentation on Java game development that included general framework information and tips and tricks on using the media APIs effectively. We also showed an application named "Ping" that demonstrated some of the ideas we discussed (see Figure 1).

Chet Haase is an engineer in the Java 2D team at Sun Microsystems, Inc., where he focuses on graphics rendering and performance issues for the Microsoft Windows platform. Catch his blogs and articles at http://javadesktop.org.

chet.haase@sun.com

Dmitri Trembovetski is an engineer in the Java 2D team at Sun Microsystems, Inc., where he focuses on graphics rendering and performance issues for the Solaris and Linux platforms. Read his frequent posts on the forums at http://javagaming.org.

dmitri.trembovetski@sun.com

Working code is a great way to illustrate an API, and we have posted the source code that accompanied our presentation on http://ping.dev.java.net. This article fully explains the ideas behind our JavaOne presentation as well as the elements of interest in the Ping application.

This two-part article is divided into three main sections:

1. Game framework
2. 2D rendering specifics
3. Ping: the demo

We will cover the first topic here and save the rest for Part 2.

Game Framework

Usually a game will want to spin off a separate thread to handle all of the main work of the per-frame game loop. This doesn't mean that all processing for the entire application happens on this thread (some things, like events, are inherently processed on different threads), but that one single thread will be used to synchronize and serialize all of the actions that must occur in every frame.

For example, a simple game loop might be:

```
public class GameLoop implements Runnable {
    public void run() {
        while (true) {
            syncFrameRate();
            gatherInput();
            updateObjects();
            render();
        }
    }
}
```

This Runnable could be used when spawning a dedicated thread:

```
Thread t = new Thread(new GameLoop());
t.start();
```

…and you're off and running. Now you just have to implement the methods in run() and you're all done. Right?

Although the loop above is pretty simplistic, it covers the basics for many types of games and it's what we'll use for the purposes of our discussion. Let's see what the details are in the methods called inside that game loop.

syncFrameRate()

At the start of any frame, you should figure out the timing information. Many games have been written that assume some kind of constant frame rate, or constant movement per frame. If you actually have a constant frame rate this may work fine, but in this world of arbitrarily fast or slow computers, different performance – based on machine configuration and load and per-frame variability in terms of scene complexity and rendering time – makes it suicidal to count on a constant frame rate. Instead, your game should figure out the per-frame movement based on how much time has actually passed. For example, don't assume that your characters move one unit per frame, but rather one unit per time unit; then no matter how much time passes between one frame and the next, the movement will look constant to the user of your program.

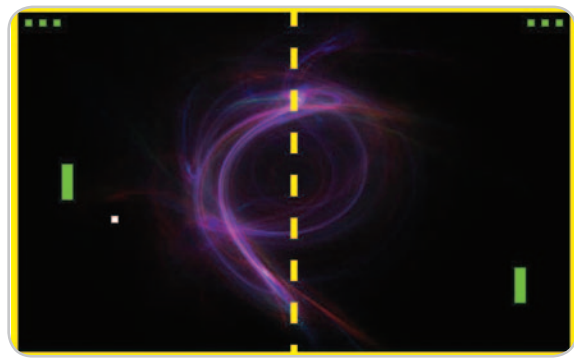


Figure 1 Ping demo: Balls, paddles; what more could anyone want? Source code available at http://ping.dev.java.net

syncFrameRate() is an attempt to boil down this idea into some simple code that does two things:

1. Tries to regulate the frame rate to some configurable speed.
2. Calculates the actual time passed since the last frame. This elapsed time will be used later in object movement calculations.

Frame Rate Consistency

Of course you could always run flat-out as fast as you can and just use the elapsed frame times to make sure the object movement is realistic. However, you may then end up spending way too much effort just getting through your game loop at that speed and not leaving enough CPU cycles for other things that your application (or your machine) might want to do instead. For example, if events are getting processed asynchronously, you want to make sure that the thread that processes those events gets enough time to actually run effectively. Also, your game may use extra cycles in a different thread to run the AI engine, the physics engine, the audio, or a host of other things. It's far better to determine a "good" or "good enough" frame rate and try to match that than to just run as fast as you can.

Another reason for consistency is that you don't want disturbing artifacts if your game runs like a demon for several frames and then crawls for a frame or two because some other thread is playing catch up. Far better to set a consistent rate lower than the peak so that the game can run smoothly at all times.

One simple way to get a consistent frame rate is to figure out what rate you would like to run at (based on preset values, user input, or timing tests before or during the game) and to throttle the game loop down to that speed during syncFrameRate().

Elapsed Frame Time

Once you've throttled the frame rate to some reasonable level, you also want to measure the exact time that has passed since the last frame. This is what you should use to calculate things like object movement – you may be close to some ideal time value by trying to set a consistent frame rate but chances are you'll usually vary slightly from that ideal (and you may occasionally vary widely due to some hiccup in performance). You should therefore ensure that your movement calculations are correct with respect to actual time and not the ideal time you are aiming for with the frame rate.

To achieve this you need to record the actual time elapsed since the last time around the game loop. Later movement calculations will then use this elapsed time value to come up with the time-correct values.

syncFrameRate() Code

The following code makes sure that your game is not trying to run any faster than this rate (the nsPerFrame variable):

```
 1:    public void syncFrameRate() {
 2:        long nextFrameTime = prevFrameTime + nsPerFrame;
 3:        long currTime = System.nanoTime();
 4:        while (currTime < nextFrameTime) {
 5:            Thread.yield();
 6:            try {
 7:                Thread.sleep(1);
 8:            } catch (Exception e) {}
 9:                currTime = System.nanoTime();};
10:        }
11:        elapsedFrameTime = currTime - prevFrameTime;
12:        prevFrameTime = currTime;
13:    }
```

Let's analyze some of the methods used in the code.

Line 3: System.nanoTime() is a new method in JDK 5.0 that finally enables the use of high-resolution timers in the Java core classes. The old System.currentTimeMillis() call that applications typically use is sufficient for most static or low frame-rate situations, but games usually require a finer degree of control than higher-resolution timers can provide. In recent simple timing tests on Windows machines I found System.currentTimeMillis() to have a minimum resolution of 10–15 milliseconds, whereas System.nanoTime() had a much better resolution of only one millisecond or so.

Line 5: Thread.yield() ensures that we cede CPU time to other waiting threads every time around the timing synchronization loop, even when we have exceeded our nsPerFrame time; sharing the CPU with other threads is usually a good idea.

Line 7: Thread.sleep(1) – in our demo, we set this call to sleep for the minimum amount of time before spinning around the loop again. You may want to set this sleep time more carefully in your game according to how much time you can afford to give away to other threads. The trade-off here is that you want to sleep for the maximum amount of time possible to allow other threads to have fair access to the CPU, but the minimum amount of time so you can still wake up soon enough to hit your target time to process the next frame; sleeping too long can make it impossible to hit your ideal frame rate.

Line 12: prevFrameTime is assumed to be a variable of type long that is retained between frames just for the purposes seen in this syncFrameRate() method.

Line 11: elapsedFrameTime is the actual time passed since the last time around the loop. It's used in later processing of things like object movement.

gatherInput()

This stage is used to process all of the "input" that has accumulated since the last frame. This input includes both local input events such as keyboard and mouse as well as networking events in networked games.

Input is actually a two-stage process, the second of which is the call to gatherInput(). The first half of the input process occurs before this step on other threads. We'll go through both of these input stages below: Event accumulation and Intelligent Event accumulation.

Important Caveats and Petty Justifications

We (the authors) are not game programmers, instead we develop graphics libraries and tend to know a thing or two about game programming concepts. Therefore this article is intended more for programmers who would like to know more about how to write basic games in Java rather than advanced game programmers. We hope the information about Java programming will benefit anyone who wants to know more about doing graphics programming on the Java platform, but we don't want to mislead anyone who is looking for state-of-the-art information on how to write a really slick circa-1980's arcade game...

Demo

This demo illustrates the points of the article and presentation and is not intended to be the end-all of 2D game programming. Instead, we focused on the general concepts of game frameworks and Java2D tips and tricks. So take the game in the spirit in which it is offered, consume it, and have fun writing something better.

Event Accumulation

In a typical AWT/Swing application, events are processed on the Event Dispatch Thread by calls to EventListener objects whenever events occur. The same thing happens here – we will attach EventListeners to find out when specific events occur. The only difference is that we won't process the events until the later gatherInput() stage. Instead we'll simply accumulate the information about what has happened and wait until gatherInput() to actually do something about all of those events.

The process for listening to events is the typical AWT/Swing one. First register some object to be a listener:

```
EventListener eventGatherer = new EventGatherer();
frame.addKeyboardListener(eventGatherer);
```

The actual process of gathering the input happens in the EventGatherer class that we have created. In the class below, we're assuming that spacebar events are of interest. Note that an actual implementation of the class would need to override the other KeyboardListener events as well (and may have many other methods besides); we are concentrating only on showing the process of tracking these types of events:

```
public class EventGatherer implements KeyboardListener {
    public void keyPressed(KeyEvent e) {
        if (e.getCharCode == KeyEvent.VK_SPACE) {
            spacebarEvents.add(e);
            return;
        } else if (...) {
            // etc.
        }
    }
}
```

Intelligent Event Accumulation

In the EventGatherer code above, note that we are calling spacebarEvents.add(e) instead of just toggling a variable that determines whether the spacebar is being held or not. In general, we want to accumulate information about all of the relevant events that occurred in each frame, not just which state things are in at the end of a frame.

As an example, the spacebar might be responsible for shooting some rapid-fire projectile. It's not enough to know whether the key is down or up once per frame because the time elapsed between frames may be long enough to allow several shots to be fired in that time. Instead, we need to know how long the spacebar has been down. Or, if it was pressed and released all within the same frame, we need to know how long it was down before it was released. Furthermore, if it was pressed and released several times, we should track that total time held and perhaps the number of times it was pressed and released.

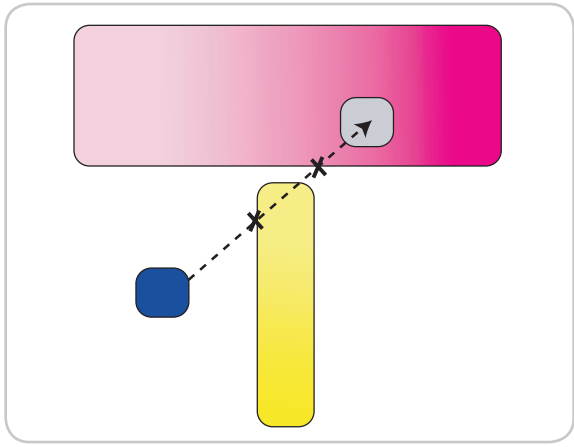


Figure 2 Collision detection finds two possible collisions this frame

We may want even more information than the total time held. Say a key determines movement for a character; the longer the key is down the more the character moves. If we track total time held, that at least gives us constant movement per time unit the key is held, but it will make character movement very stilted as characters will always move at the same rate. A more natural motion is to use acceleration and perhaps deceleration, taking the time held into account when determining how fast to move the character.

For all of these additional factors, we need to record additional information about each event. It's not enough to simply record that an event happened. Instead, we need to record when that event occurred, as well as what that event was.

The simple call to spacebarEvents.add(e) above hints at this; we are adding this new event to some list of events for that key, but we are adding it with the full event structure so we can record the event times as well as the event actions. From this information, we can later decode the information we need to track such things as total hold times and press/release data.

gatherInput(): Processing the Input

The steps outlined above (accumulating the input data) happen throughout a frame. Later on, exactly once per frame, the application takes all of the accumulated data and processes it accordingly:

```
public void gatherInput() {
    if (spacebarEvents.size() > 0) {
        processSpacebarEvents(elapsedFrameTime);
    }
    // etc.: process all other events here as well
}
```

It's important to synchronize access to the structure used for storing the events. Since we are, by definition, accessing that structure from different threads, the program needs to ensure that it only accesses it from one thread at any given time. Moreover, since we want to process all events at one time in gatherInput(), no more events can be added to the queue in parallel while we are working on it. Failure to do this will mean we process events that have not actually occurred yet. This is because any frame is being processed based on the time just passed, and we cannot take into account things that are happening while we are processing that frame because all of that must be processed on the next frame.

The method processSpacebarEvents(long) is passed in the elapsed time that we calculated in syncFrameRate(), and uses the argument to calculate event actions based on the events that occurred for the spacebar as well as the times that those events occurred.

This may appear pretty sketchy, but that's what having sample code is all about. In the Ping demo we do the event accumulation and processing steps outlined above to calculate accelerating movement for the game paddles in that code.

What About Networking?

At a high level, network events can be handled exactly as described earlier for local events – you accumulate the network events in separate threads asynchronously and then, once per frame, you process these events and calculate any necessary movement and actions.

The details behind this high-level view are, of course, much trickier. Networking can be far more complicated than the simple local event processing described earlier. We decided not to pursue networking in this article and demo and to instead just focus on simple game frameworks and rendering issues. Nevertheless it's useful to at least call out some of the important issues that will arise in any networked game.

Network Events

There are many different types of networking events that you may want or need to process. These include player movement, player action, and synchronization:

- ***Player movement:*** Players on different machines will presumably have characters they control locally whose positions need to be communicated to the other machines involved in the game.
- ***Player action:*** Actions such as firing, colliding, and speaking need to be communicated to all appropriate clients in the game.
- ***Synchronization:*** Timing synchronization is critical in networking. Chances are slim that games on separate clients began at exactly the same moment, or even that the clocks on the different game clients are matched. There needs to be some kind of handshaking and potentially ongoing sanity-checking to make sure that one client's view of time and the world is the same as all other clients.

Networking Considerations

There are a variety of important considerations that must be taken into account when designing and implementing a network architecture for a game including network architecture, networking artifacts, and protocol:

- ***Network architecture:*** Your game could use a peer-to-peer architecture, where each client of the game speaks directly to the other clients. This is possibly a simpler architecture with lower overhead, but requires that each client keeps a more complete model of the universe locally. This approach also has security implications, as any client has the ability to hack the universe and thus cheat in the game. A client/server architecture has higher overhead, but benefits from the security and manageability standpoint because there is always one single machine that has the master view of the universe.
- ***Networking artifacts:*** Even in these days of widespread broadband access, there are still important network artifacts that cannot be ignored such as latency, synchronization, and loss.

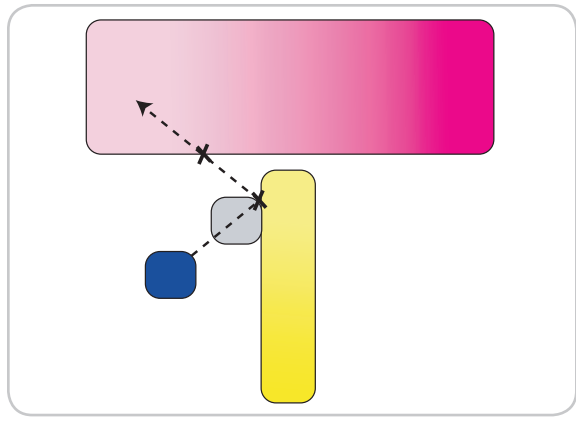


Figure 3 Ball bounces due to first collision and we recalculate for new trajectory, finding a new collision with the wall

- ***Latency:*** Even if all players have DSL, there are probably still inherent delays in transmission between the clients, more so with geographically widespread clients. Your game must deal with latency aggressively through things like movement prediction. For example, instead of depending on every client transmitting new positions for their characters every frame, it would be far better for clients to simply transmit when movement trajectory/velocity has changed and what those new values are. Then the other clients can do their own calculations of where that character is every frame until they receive new information from that client.
- ***Synchronization:*** As mentioned earlier, all of the clients need to have the same view of time in order to have the same view of the game universe. Issues such as latency can make synchronization difficult (you cannot tell the machines to start an action at the same time if the machines don't get that message to start at the same time).
- ***Loss:*** There is always the potential for data loss over the network. If your game is dependent upon every data packet reaching its destination intact, there will be problems when that doesn't occur.
- ***Protocol:*** The issue of data loss over the network raises the important issue of networking protocols. There are various protocols to choose from, some of which handle such things as packet loss. These higher-level protocols can detect loss and retransmit lost data, but at the cost of a higher overhead protocol with longer transmission and processing times. In general, with the very fast networking requirements of games, a better choice is the smallest and cheapest (in terms of overhead) protocol available. But if this choice means that your game may experience data loss and out-of-order data packets, your code must account for that possibility.

updateObjects()

Once we have determined our elapsed frame time and processed all of the input for this frame, it's time to update the positions of all objects. Object positions are based, in general, upon two factors: elapsed time (a moving object must be moving at some rate per time) and change in movement (presumably triggered by some event). There is also object collision that can cause movement change, but we'll handle that as part of this updateObjects() process.

The general idea behind updateObjects() is simple: for all objects that can move this frame, calculate their movement and reposition them accordingly:

```
public void updateObjects() {
    for (Movable movable : movables) {
        movable.update(elapsedFrameTime);
    }
    processCollisions();
}
```

The interesting for() statement is based on the new, expanded for() semantics in release 5.0. Essentially, we're just saying that for all of the objects that can move, go ahead and move them by calling update() on those objects.

The update() method left unspecified above is a method that takes the current elapsed time for this frame and calculates the new position for that object according to movement information that the object knows about. This includes the object's previous position and its speed (or acceleration, in the case of nonlinear movement).

processCollisions()

Collision processing can be one of the most complex parts of your game depending on several factors: how many objects can potentially collide; how many of those objects are dynamic (moving) objects; and how physically correct you would like your collisions to be.

In the simple case of the Ping demo there are only two types of collisions possible: the paddles with the walls, and the ball with the paddles and walls. The paddle/wall collisions are handled implicitly during the gatherInput() stage; keyboard events determine paddle movement and this movement is merely constrained to be within the area defined by the walls. The ball collisions are handled during the processCollisions() call and consist of checking the ball boundaries against the wall/paddle boundaries and calculating new positions based on any collisions and bounce effects.

Interestingly, even this "simple" collision caused artifacts in the original version of the Ping demo to use only the ball center instead of the ball bounds, as our code simplified the collision detection algorithm. This meant that the ball would not "collide" at exactly the right time and place; if one part of the ball collided with a paddle but the center of the ball did not, we would not detect the collision and the ball would flow right through. The current version of the demo fixes that problem by using simplifying assumptions about which sides of the ball can collide with which sides of each paddle.

In any case, the processCollisions() step in our demo consisted of checking a single moving object (the ball) against the various elements in the scene with which it might collide. Note that multiple collisions might be detected (the ball trajectory might intersect both a paddle and a wall, for example); in that case, the closest collision would be selected (if the ball hit the paddle first, the collision against the wall would be invalid because the ball would bounce off the paddle). If a collision was detected, a bounce would occur and a new position would be calculated. Then collision processing must happen again for the new trajectory that the ball takes after that first bounce. This process must be repeated until there are no more collisions for that object in this frame.

This process is illustrated in Figures 2–5.

Figure 2 shows the ball (blue) traveling toward the paddle (yellow) with the trajectory illustrated by the dotted line, in the direction of the arrow. We calculate two collisions (black Xs), one with the paddle and one with the wall (pink). The original endpoint of the ball would be where the gray ball is painted inside the wall (if there were no collisions).

Figure 3 shows the next step in processCollisions. We've calculated that the ball will bounce off the paddle first and travel in the resulting trajectory toward the wall, resulting in the new collision shown with the new black X.

Figure 4 shows the next step in processCollisions() where we have calculated a bounce off the wall and a trajectory for the ball flying back out into the playing area.

Finally, in Figure 5 we show the final state of the ball (still in gray) with the trajectory and collisions as shown by the lines and Xs that caused the ball to end up where it is.

One thing that this example glosses over is the actual calculations involved, both the calculation of each collision as well as the calculation of the bounce. We'll save that discussion for the code itself where you can see what we did to calculate these results in the demo. However three points are important to bring up about this process.

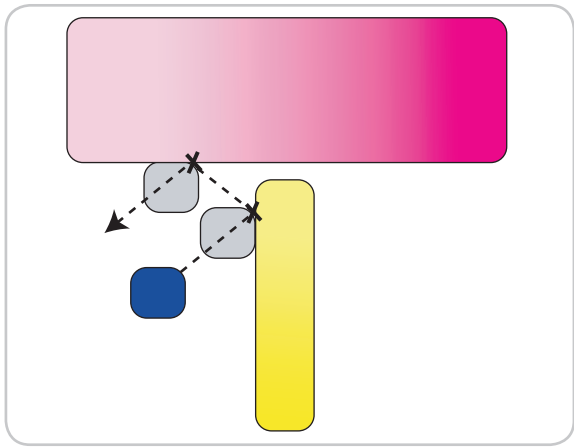


Figure 4 Ball bounces again due to second collision and we recalculate for new trajectory, finding no more collisions

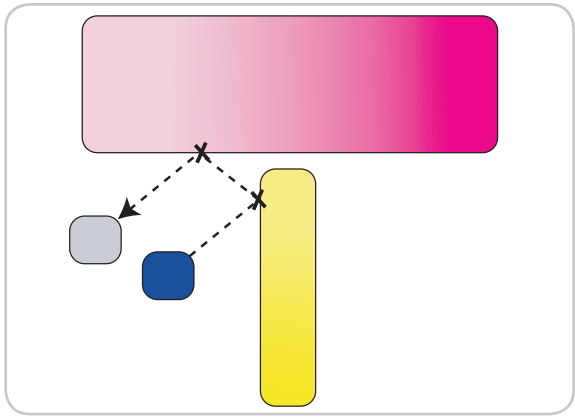


Figure 5 Final location of ball after all collision processing

1. *Trajectory Versus Position Collision*

The simplest approach to collision detection would be to simply calculate whether the end position of an object overlaps with any other object in the world. If so, do the right thing. If we had taken this approach in the previous example, it's easy to see what artifacts would result. In the first picture we would not have calculated the bounces as shown; we would have just detected that we were inside the wall and would then calculate a bounce based on that collision while ignoring the first collision that actually should have occurred with the paddle.

To do this correctly requires calculating collisions based on object trajectories, not just object positions. In the figures this means we need to calculate the collisions between the dotted line representing the object movement and the objects that the line encounters.

2. *Physically Accurate Collisions*

Figures 2–5 show a single vector that we use to base the collisions on. However, the ball occupies an area, not just a point. To calculate correct collision information, we would need to calculate the intersection of the ball area with all available objects, not just some point or line representing its trajectory.

To simplify the demo code (and make sure it was ready to be shown at JavaOne), we actually used the simplified approach of intersecting a single line in the version we showed at the conference. It worked pretty well in general, but had visible artifacts. For example, if the collision point is based on the center of the ball, you'll notice the ball passing through objects up to that center point; not very realistic (even for a game this unrealistic to begin with), and the artifacts only worsen with the size of the objects that take this simplified approach.

For a real game with much more complex objects than our simple ball, say cars or planes, you would need to decide what level of simplification you could live with, if any, and what artifacts those assumptions would cause. For example, you could use a simple bounding box or circle around an object to calculate collisions; this is better than the point-based approach above, but would still have artifacts such as detecting collisions when there were none (because the bounding area might collide when the actual object inside that area would not).

3. *Bounce Calculation*

The figures assume a standard thetaI == thetaR (angle of incidence equals angle of reflection) equation for the bounce. This works fine, in general, but ignores some more complicated and interesting possibilities such as putting english, or spin, on the ball. For example, if a paddle is moving when the ball hits it, it would be reasonable to figure in some kind of angle perturbation based on the speed of paddle movement, such as you would get if a tennis racket were moving sideways when it struck a ball (thus putting spin on the ball and changing the angle of reflection). The Ping demo shows some of this behavior, as we modify the angle of reflection based on the speed that the paddle is moving when it hits the ball.

This was the case for our simple one-collidable-object case in Ping – imagine how complex this process would be if our world consisted of multiple dynamic objects or if the objects were dynamically deforming or even if the collisions were accurately modeled on complex objects. The equations in all of these cases become much more involved and need to be carried out potentially many times in each frame as objects bounce off possibly several other objects in any given frame.

render()

The main idea in the rendering stage is to traverse the world of visible objects and draw all of them to the screen. Of course the details are a bit more involved. In general, for most 2D games, the scene rendering might be boiled down to something like:

```
1:    public void render() {
2:        renderBackground();
3:        renderObjects();
4:        renderForeground();
5:        showBuffer();
6:    }
```

Let's analyze each method call in turn.

Line 2: renderBackground() draws the background image or any other static graphics that tend to be the same in every frame. It underlies everything else in the game, so draw it first.

Line 3: renderObjects() draws all of the static and movable objects in a game. This includes things like the walls, paddles, and ball from the Ping demo. You may want to break apart the static and movable objects and draw one or the other first (for example, maybe your player characters should always appear on top of any static objects, in which case you should draw the static objects before the movable objects).

Line 4: renderForeground() draws anything that overlaps the game, such as the score, any help, status windows, or text, a dashboard in a driving game, or other things that are considered to be separate from the world of the actual game objects. Since these things need to be rendered on top of the rest of the game, we draw them last.

Line 5: showBuffer() – in general, games will always draw to an offscreen buffer instead of directly to the screen. This makes animations appear much smoother and game play more enjoyable. After everything is rendered to the back buffer, the final step is to show this buffer on the screen. The details of how to do that will be discussed in Part 2.

This is a pretty gross simplification of the rendering process. The actual work your application would do to rendering would be highly dependent upon the type of game that it is. For example, 3D games have very different requirements than 2D games. In the 2D domain, there are many different types of games (scrollers, top-down, action, word, arcade, etc.); how you choose to render your world depends much on what you have to render and what the interface is like.

There are, however, some basic elements of 2D rendering that are common to many different games in that domain. Next issue, we will dive into 2D rendering issues, such as buffering, fullscreen, image usage, and performance tips. We will also give an overview of the Ping demo code.

Industry News

JBoss Becomes a Member of the Eclipse Foundation

(*Atlanta/Ottawa*) – JBoss, Inc., the Professional Open Source company, and the Eclipse Foundation, a community committed to the implementation of a universal platform for tools integration, have announced that JBoss has become a member of the Eclipse Foundation and will contribute code toward the J2EE Standard Tools Project. By working together, the two will combine their expertise to provide a more complete and tightly integrated open source tools and middleware solution.
www.jboss.com
www.eclipse.org

Stottler Henke Introduces DataMontage Software

(*San Mateo, CA*) – Stottler Henke Associates, Inc., has announced DataMontage, a new software product that enables rapid visual analysis of complex, time-oriented data. DataMontage is a Java class library that offers flexibility to display information-dense collections of timelines, time-series graphs, and time-stamped notes. It enhances reporting and decision support systems for medical data review, manufacturing data analysis, market research, and planning and scheduling – environments in which professionals are often called upon to quickly glean meaning by detecting patterns spanning many variables.
www.stottlerhenke.com

Red Rock Powers the Department of Energy's COMcheck Software

(*Salt Lake City/Richland, WA*) – Red Rock Software and Pacific Northwest National Laboratory have released a new Java version of COMcheck-EZ – a free computer software application provided by the U.S. Department of Energy (DOE). It streamlines the energy code compliance and approval process for those in the commercial building industry. The new Java version provides platform-independent user interfaces allowing the software to run on either Windows or Macintosh computers.
www.redrocksw.com

AMX Extends NetLinx with Java for Industry's First Dual Language Control System

(*Richardson, TX*) – Demonstrating the company's support of open standards, AMX Corporation has announced Duet, an extension of AMX's NetLinx platform that provides dealers with the opportunity to integrate AMX's advanced control systems using the standards-based Java programming language. By providing the option to program using NetLinx, Java, or both, Duet expands the development opportunities for existing NetLinx programmers while opening the door to 4.2 million Java programmers who now can develop innovative applications for control systems.
www.amx.com

Borland and Vignette to Help Enterprise Software Teams

(*San Jose, CA*) – Borland Software Corporation and Vignette Corp. have announced a relationship and combined solution designed to help organizations accelerate the creation of portlets with custom functionality, improve the design and quality of Web interfaces, and consolidate management of disparate portlet functionality across the enterprise. The solution will consist of a portlet creation plug-in for Borland JBuilder 2005 and the Vignette Application Portal Borland Developer Edition. This wizard-based solution is designed to help developers create custom portlets based on the Java standard (JSR 168) to deliver security features inherent in the J2EE environment, shorten development time, and enhance scalability and performance for enterprise applications.
www.borland.com
www.vignette.com

Jinfonet Software Awarded GSA Schedule

(*Rockville, MD*) – Jinfonet Software has announced that JReport is now listed on the United States GSA Schedule. Also referred to as Federal Supply Schedules, GSA Schedules provide government agencies with access to high-quality products and services at the most-favored customer price.

JReport is a 100% J2EE reporting solution that seamlessly embeds into any application with minimal development effort, resulting in a fast time to deployment and enhanced application value. By combining enterprise and ad hoc reporting with analysis, JReport 7 provides users with the ability to generate, share, and analyze on-demand or scheduled reports from within any application. JReport's GSA schedule number is GS-35F-0702P.
www.jinfonet.com

Lockheed Martin Unit Selects Parasoft's Jtest, C++Test, and Insure++

(*Monrovia, CA*) – Parasoft, a provider of automated software error detection and reporting solutions, announced that Lockheed Martin Maritime Systems & Sensors (MS2) has selected Parasoft's Jtest, C++Test, and Insure++ to support quality testing for its software.

Jtest and C++Test provide automated unit testing and coding standard analysis for Java and C/C++ applications by automatically verifying compliance to hundreds of coding rules while automatically generating and executing unit tests in order to ensure quality early on in the software development life cycle. Insure++ detects elusive memory errors, such as corruption, leaks, and allocation errors in C/C++ code. Parasoft's automated error detection solutions enable software development organizations to improve development effectiveness, reduce cost, and increase software quality.
www.parasoft.com

Dralasoft Launches Aggressive Channel Reseller Program

(*Westminster, CO*) – Dralasoft, Inc., a provider of business process management (BPM) solutions, has announced the debut of a results-oriented Channel Reseller Program aimed at attracting top-name VARs (Value-Added Resellers) in a broad range of vertical markets. Dralasoft believes its Channel Reseller Program will provide partners with the strongest Business Process Management product capability in the industry.

Dralasoft's new program will place heavy emphasis on personal contact, training, and ongoing exchange of information. Newly signed partners will not only attend a quick-start program orientation, but also will receive comprehensive product and sales training at the Dralasoft facilities. These initial events, which lead to "Authorized" reseller status, will be followed by shared marketing and sales activities.
www.dralasoft.com

Reach Over 100,000

Enterprise Development Managers & Decision Makers with...

Offering leading software, services, and hardware vendors an opportunity to ***speak to over 100,000 purchasing decision makers*** *about their products, the enterprise IT marketplace, and emerging trends critical to developers, programmers, and IT management*

Don't Miss Your Opportunity to Be a Part of the Next Issue!

Get Listed as a Top 20* Solutions Provider

For Advertising Details Call 201 802-3021 Today!

*ONLY 20 ADVERTISERS WILL BE DISPLAYED. FIRST COME FIRST SERVE.

SYS-CON MEDIA

The World's Leading i-Technology Publisher

From Within the Java Community Process Program

The 'wired' and 'wireless' stacks

Onno Kluyt

Welcome to the October edition of the JCP column! Each month you can read about the Java Community Process: newly submitted JSRs, new draft specs, Java APIs that were finalized, and other news from the JCP. This month there are no less than 22 JSRs worthy of discussion.

J2SE 5.0 Platform Approved

The J2SE 5.0 technology, also known by its development codename "Tiger," is represented by 12 different JSRs including the so-called umbrella JSR 176, which coordinates the coming together of the individual technology JSRs. The JSRs successfully passed their Final Approval Ballot. By the time you read this column the final release should be available from java.sun.com.

The "Wired" Stack of the J2ME Technology

The J2ME environment informally recognizes two technology stacks: the CDC and Personal Profile combination (sometimes referred to as the "wired stack") and the CLDC and MIDP combination (sometimes referred to as the "wireless stack"). A set of four JSRs related to the first stack is progressing through. JSRs 216 and 217 define version 1.1 of the Personal Profile and Personal Basis Profile, respectively. Both JSRs update the J2ME technology with changes from the J2SE 1.4 platform (e.g., updates to AWT and JavaBeans APIs). JSR 218 is developing version 1.1 of the Connected Device Configuration (CDC) and JSR 219 is creating an update to the J2ME Foundation Profile. All four JSRs have now entered Public Review. These JSRs progress under JCP 2.6 and thus the last week of the review will give the ME EC another opportunity to vote on these JSRs.

There is one more JSR in the J2ME space that I'd like to call your attention to. This is Siemens' and Motorola's Mobile Telephony API JSR proposal, number 253. An example of the use of this JSR would be applications that provide conferencing, call scheduling, and voice services, but with the limited resources of the devices in mind.

JavaServer Faces Starts Work on Its Second Installment

JSR 252 was approved by the SE/EE EC with its main goal of providing alignment for JavaServer Faces with JSR 245, which is developing a new version of the JavaServer Pages technology.

Related to the enterprise space is JSR 181, which BEA as the spec lead has succeeded in advancing into Public Review. This JSR defines annotations and metadata for the enterprise Web services sector and builds upon JSR 174, which lays the foundation for annotations functionality in the Java platform.

OSS/J News

Nakina Systems has proposed to develop a specification for an OSS Discovery API (JSR 254). This API would provide mechanisms for the discovery of physical devices and their associated data present on a network. Some areas that the JSR will focus on are discovery seed data, agents, concurrency control, and device security credentials. As the title of the JSR suggests, it's part of the OSS through Java Initiative (OSS/J). A good place to find "OSS/J"-minded developers is at its community on java.net.

And Finally, JSR 255 JMX 2.0

This JSR submission comes ahead of the umbrella JSR for the J2SE 6.0 platform. It will make modifications to the Java Management Extensions as well as the JMX Remote API. Ease of use is one focus of this effort, and the JSR will work on some new functionality.

It will look at generics, use annotations to make writing MBeans easier, address interoperability and versioning concerns for Open MBeans, and provide support for cascaded MBean Servers.

That's it for this month. I'm very interested in your feedback. Please e-mail me with your comments, questions, and suggestions.

Onno Kluyt is the director of the JCP Program Management Office, Sun Microsystems.

onno@jcp.org

> "The J2ME environment informally recognizes two technology stacks: the CDC and Personal Profile combination and the CLDC and MIDP combination"

JAVA DEVELOPER'S JOURNAL Advertiser Index

Advertiser	URL	Phone	Page
Altova	www.altova.com	978-816-1600	13
Axalto	www.simagine.axalto.com		47
Borland	www.go.borland.com/j6	831-431-1000	9
Business Objects	www.businessobjects.com/dev/p7	888-333-6007	23
DataDirect	www.datadirect.com/jdj	800-876-3101	7
Dice	www.dice.com	877-386-3323	35
Google	www.google.com/cacm	650-623-4000	31
H&W Computer Systems	www.hwcs.com/jdj04.asp	800-338-6692	51
ILOG	jviews-info-kit.ilog.com	800-for-ILOG	17
InetSoft	www.inetsoft.com/jdj		33
InferData	www.inferdata.com/jdj	888-211-3421	41
InterSystems	www.intersystems.com/match3	617-621-0600	15
IT Solutions Guide	www.sys-con.com/itsg	888-303-5282	59
M7	www.m7.com/d7.do	866-770-9770	27
Microsoft	msdn.microsoft.com/visual/think		11
Mindreef	www.mindreef.com	+1 603 465-2204	43
Northwoods Software Corp.	www.nwoods.com/go	800-434-9820	49
Oak Grove Systems	www.oakgrovesystems.com	818-440-1234	45
Oracle	www.oracle.com/portal	800-633-0759	Cover II
Parasoft Corporation	www.parasoft.com/jtest	888-305-0041	21
Quest Software, Inc.	http://www.quest.com/jdj	800-663-4723	Cover IV
ReportingEngines	www.reportingengines.com/download/f1ere.jsp	888-884-8665	25
Scientific Toolworks, Inc.	www.scitools.com		37
Sleepycat Software	www.sleepycat.com/bdbje	510-597-2128	29
Software FX	www.softwarefx.com	800-392-4278	Cover III
Stylus Studio	www.stylusstudio.com	781-280-4488	4
Web Services Edge 2005 East	www.sys-con.com/edge	201-802-3069	61

General Conditions: The Publisher reserves the right to refuse any advertising not meeting the standards that are set to protect the high editorial quality of ***Java Developer's Journal***. All advertising is subject to approval by the Publisher. The Publisher assumes no liability for any costs or damages incurred if for any reason the Publisher fails to publish an advertisement. In no event shall the Publisher be liable for any costs or damages in excess of the cost of the advertisement as a result of a mistake in the advertisement or for any other reason. The Advertiser is fully responsible for all financial liability and terms of the contract executed by the agents or agencies who are acting on behalf of the Advertiser. Conditions set in this document (except the rates) are subject to change by the Publisher without notice. No conditions other than those set forth in this "General Conditions Document" shall be binding upon the Publisher. Advertisers (and their agencies) are fully responsible for the content of their advertisements printed in ***Java Developer's Journal***. Advertisements are to be printed at the discretion of the Publisher. This discretion includes the positioning of the advertisement, except for "preferred positions" described in the rate table. Cancellations and changes to advertisements must be made in writing before the closing date. "Publisher" in this "General Conditions Document" refers to SYS-CON Publications, Inc.

This index is provided as an additional service to our readers. The publisher does not assume any liability for errors or omissions.

Java Certification and I

by William Knight

Is the Java certification program offered by Sun really the route to a higher salary and better quality of code for businesses? I have my doubts.

In my fifth year of Java programming, after being involved in several distributed developments for large companies, a prospective employer tested my ability. "Don't worry," he said, "you'll have no trouble; this is for beginners."

Unfortunately I failed and was embarrassed – it was a dismal score. He asked me why I had scored so low and with my career and self-belief crushed and broken and in ruins on the grey carpet-tiles, I mumbled something about IDEs, esoteric details of thread control, and tricky default behavior. I dashed from the building and imagined him later berating the agent for sending somebody so obviously lying about his experience.

But I wasn't lying. For 10 years I hadn't created a source file or method name from scratch – the IDE had done it for me. I never had cause to use Java threads, and I avoided the use of default behavior on principle. When I first began programming in Java, integrated development environments already existed that created the files and provided all the correct packages and imports.

Never once was I concerned with the correct syntax for the main() method or the Java source file. What mattered to me was creating an understandable design and hitting the business goals. As a team leader, if one of my coders had written examples like the ones I'd been tested on, I'd have sent him or her on a training course to learn the language.

I later discovered that the interview questions had come from the Sun Java certification. While finding out more about certification I came across the following quotes on the Web:

"I am a Java Certified Programmer and the only thing I've gotten out of it is the ability to say that it's been exactly useless to me."

"I had hoped that certification might carry some clout, but it isn't turning out that they have as much clout as the certification vendors would have you believe."

"I'm an SCJP and SCJD. They're absolutely useless in your career."

This attitude of irrelevance is confirmed by the statistics on JobServe (the largest job database for IT personnel in the UK: www.Jobserve.co.uk). A search for Java returned 1,498 current jobs, while a search for Java and Certified returned just 10 current jobs. About 0.7% of positions ask for certification.

I decided to take the test despite the evidence and discovered that learning Java for the real world and learning it for the exam are not the same. The exam curriculum is well intended but seems to encourage dubious practices that only an experienced programmer would know about. After passing with 80%, I was left wondering why Sun had created such devious questions. My rule of thumb is that if you can't comprehend it immediately, it's wrong. Many of the questions in the exam require careful study – the exact opposite of what we, as programmers, should be trying to achieve.

Instead I would like to see the exam concentrate on the process of programming rather than the code itself; to me you can be a first-class developer, without knowing all the grammatical details, by concentrating on clarity. Businesses require the production of quality, maintainable code in a timely manner, and this can be tested. For example, the following subjects should be covered:

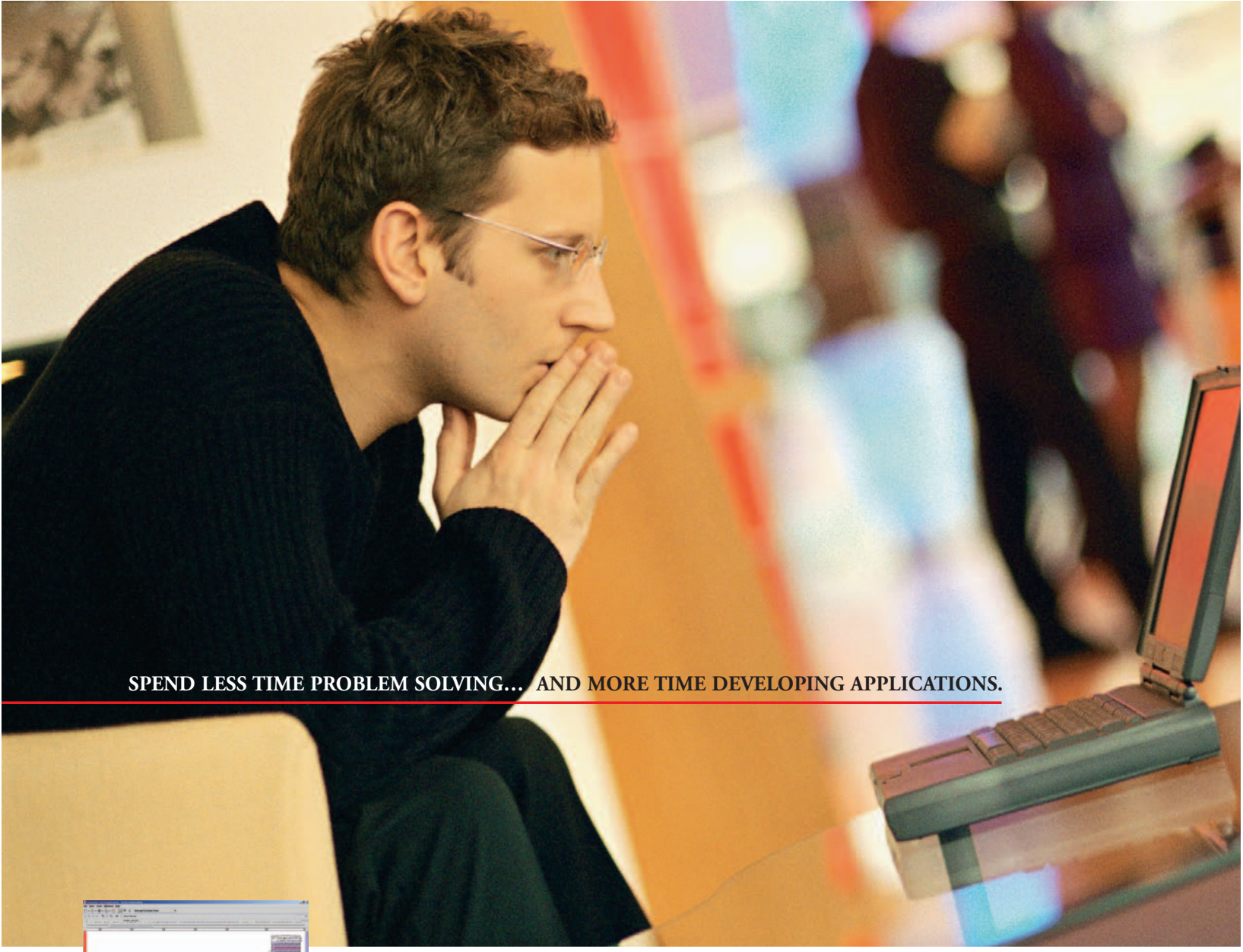
1. Comment this section of code.
2. Rewrite this code to be self-documenting.
3. How to avoid threading. The proliferation of unnecessary threading is a personal irritation. I went through 10 years of development on huge systems without needing to create threads, and now they appear everywhere.
4. What is wrong with this dry run?
5. Write the code this flow chart represents.
6. Provide suitable names for these classes.

When considering the testing of ability, it is important to remember that programming does not exist in isolation but side by side with business goals: deadlines, quality controls, and maintenance. It's not enough for a programmer to know the language – indeed I argue that in some heads, knowing the language too well can even be a hindrance to readable code. Yet the business aspects of development are entirely missing from the programming exam. It does not test language so much as it tests grammar, but the two are very different.

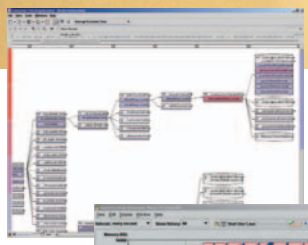
You can learn more about the Java certification program at http://training.sun.com/US/certification/java/index.html.

William Knight has been developing software for 20 years and writing about it for two. His career has taken him around the world twice and onto the staff of a selection of the world's largest companies. He currently lives and works in Devon in the UK.

william@wordsafari.co.uk



SPEND LESS TIME PROBLEM SOLVING... AND MORE TIME DEVELOPING APPLICATIONS.



PerformaSure – a system-wide performance diagnostic tool for multi-tiered J2EE applications running in test or production environments.

JProbe – a performance tuning toolkit for Java developers.

Join The Thousands of Companies Improving Java Application Performance with Quest Software.

Whether it's a memory leak or other performance issues, Quest Software's award-winning Java products — including JProbe® and PerformaSure™ — help you spend less time trouble-shooting and more time on the things that matter. Quest's Java tools will identify and diagnose a problem all the way down to the line of code, so you no longer have to waste time pointing fingers or guessing where the problem lies. Maximize your team's productivity with Quest Software by downloading a free eval today from **http://www.quest.com/jdj**.

QUEST SOFTWARE®