

JAVA™ DEVELOPERS JOURNAL

JavaDevelopersJournal.com

Volume: 3 Issue: 9 1998

I Told You So
by Sean Rhody pg. 5

Java: The Smarter Choice?
by Jim Redman pg. 7

Widget Factory JWizard
by Claude Duguay pg.22

Straight Talking The Land of the Rising Sun
by Alan Williamson pg.27

Product Reviews MindQ Java
by David Jung pg.30

Java Studio
by Dana Crenshaw pg.34

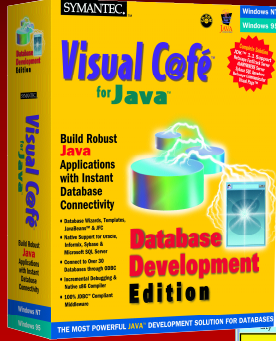
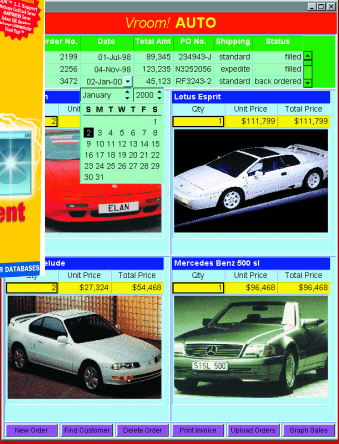
Optimize it!
by Achut Reddy pg.43

ROAD:BeanBox
by Ed Zebrowski pg.54

The Grind The Application Server Gold Rush
by Java George pg.66

CASE STUDY

Xerox Success

KL Group LiveTable

Cover Story: Collections in JDK Bhaven Shah
New API framework will change the shape of Java's data structure 8

JDJ Feature: Enterprise JavaBeans Liane Acker
Increasing portability between servers after development 16

Case Study: Xerox Uses Java to Cut Development Time in Half Cara O'Sullivan
Visual Café smooths the transition to Java 38

Java APIs and Products for Consumer Devices Ajit Sagar
Providing a ubiquitous platform and language for communication 46



CORBACorner: IIOP Explained Michael Barlotta
It may be the next universal Internet protocol 50



Applet and Servlet Communication Chad Darby
Providing a new way to develop server-side solutions 56



Full Page Ad

Full Page Ad

Full Page Ad

EDITORIAL ADVISORY BOARD

Ted Coombs, Bill Dunlap, David Gee, Arthur van Hoff, Brian Maso, Miko Matsumura Kim Polese, Sean Rhody, Rick Ross, Richard Soley, George Paolini

Editor-in-Chief: Sean Rhody

Art Director: Jim Morgan

Executive Editor: Scott Davison

Managing Editor: Anita Hartzfeld

Senior Editor: M'lou Pinkham

Editorial Assistant: Brian Christensen

Technical Editor: Bahadır Karur

Visual J++ Editor: Ed Zebrowski

Visual Café Pro Editor: Alan Williamson

Product Review Editor: Jim Mathis

Games & Graphics Editor: Eric Ries

Tips & Techniques Editor: Brian Maso

WRITERS IN THIS ISSUE

Andrei Cioroianu, Scott Davison, Claude Duguay, George Kassabgi, Pascal Ledru, Jim Mathis, Harlan McGhan, Lynn Monson, Jim Redman, David Reilly, Sean Rhody, Ajit Sagar, Alan Williamson

SUBSCRIPTIONS

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

Subscription Hotline: 800 513-7111

Cover Price: \$4.99/issue.

Domestic: \$49/yr. (12 issues) Canada/Mexico: \$69/yr. Overseas: Basic subscription price plus air-mail postage (U.S. Banks or Money Orders). Back Issues: \$12 each

Publisher, President and CEO: Fuat A. Kircaali

Vice President, Production: Jim Morgan

Vice President, Marketing: Carmen Gonzalez

Advertising Manager: Claudia Jung

Advertising Assistants: Robin Forma

Jaclyn Redmond

Accounting: Ignacio Arellano

Graphic Designers: Robin Groves

Alex Botero

Webmaster: Robert Diamond

Senior Web Designer: Corey Low

Customer Service: Sian O'Gorman

Paula Horowitz

Online Customer Service: Mitchell Lowe

Customer Service Inters: Angela Frasco

Ann Marie Millio

EDITORIAL OFFICES

SYS-CON Publications, Inc.

39 E. Central Ave., Pearl River, NY 10965

Telephone: 914 735-1900 Fax: 914 735-3922

Subscribe@SYS-CON.com

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

POSTMASTER: Send address changes to:

JAVA DEVELOPER'S JOURNAL, SYS-CON Publications, Inc., 39 E. Central Ave., Pearl River, NY 10965-2306.

© COPYRIGHT

Copyright © 1998 by SYS-CON Publications, Inc. All rights reserved. No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopy or any information storage and retrieval system, without written permission. For promotional reprints, contact reprint coordinator: SYS-CON Publications, Inc. reserves the right to revise, republish and authorize its readers to use the articles submitted for publication.

ISSN # 1087-6944

Worldwide Distribution by Curtis Circulation Company

739 River Road, New Milford NJ 07646-3048 Phone: 201 634-7400

Java and Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

SYS-CON Publications, Inc. is independent of Sun Microsystems, Inc.

SYS-CON PUBLICATIONS

All brand and product

FROM THE EDITOR

Sean Rhody, Editor-in-Chief



I Told You So

About two years ago a colleague of mine named Joe leaned over my cubicle wall and said, "Hey, I just downloaded this new language called Java. It's pretty cool!" At the time I can't remember being very excited about another programming language. I was a PowerBuilder maven and Joe was up to his eyeballs in C++. That probably accounts for some of my disinterest and Joe's initial drooling (sorry, Joe, but you did). Two years and one large-scale Java project later, I'm as much a convert as Joe.

That doesn't mean I want to rebuild everything that's ever been written in Java, nor does it mean I think PowerBuilder's obsolete (or C++ for that matter). I recently attended a conference where a technical representative from Sun was discussing Java for the enterprise. He asked, "Where should we use Java?" The answer was most appropriate: "Where you need it."

There's nothing a client hates more than an "it depends" answer. Unfortunately, it's often the truth. So it is with this answer. Where you're going to use Java depends on your needs and strategic direction. Should you use Java everywhere? Almost without a doubt, no. There are things Java is good at and things Java is not good at. There are also practical considerations, such as corporate infrastructure, that have nothing to do with Java's capabilities but impact where Java is and is not practical.

As an example of what's not practical, look at Corel's attempt to re-create its office suite in Java. In theory, Java is as suited for this as any language, more so than some with strong multitasking. But this was not the right place for Java. For one thing, you need every ounce of speed on a machine to make these overprogrammed suites perform well. JIT compilers notwithstanding, native code is still faster right now.

Even worse, you know someone would get the bright idea to host this in a browser. Why buy a thousand copies when you can access a single copy over the LAN? It'd be a tossup as to who would shoot that guy first - the network administrators who

were dealing with network overload or the users who were waiting hours for their new, "improved" software to load.

Probably the biggest lesson that needs to be learned is that Java is part of an architecture, not an architecture unto itself. I hear companies saying, "We've got to go to Java," and I can understand their frustration and desire. The Internet has turned the safe, known world of client/server on its ear, and the closest thing to a standard that most of us can find is Java.

That's great. I'm all for Java being the language of the Net. It's compact, it's elegant and it's fun to program in. The problem is that you can't simply swap Java for whatever language you've been doing two-tier development in and expect to have a solution. For one thing, JDBC is still not as far along as ODBC or native drivers. For another, it's harder to provide the same rich GUI, at least on Windows platforms. Love it or hate it, Windows is still the overwhelming desktop today, and we need to be able to build better looking Java apps if Java is to become a dominant force on those desktops. Some of this is due to the browsers rather than to the language itself. I have to applaud the people who put HTML together as a document language, but as an application environment it leaves a lot to be desired.

So what do we do? It's pretty simple really. We need to put Java where it belongs. It's not the only tool we have, and we must have good reasons for selecting it over other languages and products. At the same time we need to push for improvements in the browsers and compilers, and hope that a JavaOS will actually make sense, both from a programmatic and, in an era of \$700 PCs, an economic sense. Meanwhile, I need to call Joe and tell him he was right. I hope he doesn't rub it in. ☘

About the Author

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

Full Ad

SYS-CON Publications CONTACT ESSENTIALS

CALL FOR SUBSCRIPTIONS

1 800 513-7111

International Subscriptions
& Customer Service Inquiries
914 735-1900

or by fax: 914 735-3922

E-Mail: Subscribe@SYS-CON.com
<http://www.SYS-CON.com>

MAIL All Subscription Orders or
Customer Service Inquiries to:

DEVELOPER'S JAVA JOURNAL

Java Developer's Journal

<http://www.JavaDeveloperJournal.com>

DEVELOPER'S PowerBuilder Journal

PowerBuilder Developer's Journal

<http://www.PowerBuilderJournal.com>

DEVELOPER'S COLD FUSION JOURNAL

Cold Fusion Developer's Journal

<http://www.ColdFusionJournal.com>

DEVELOPER'S VRML JOURNAL

VRML Developer's Journal

<http://www.VRMLDevelopersJournal.com>

POWERBUILDER 6.0

Secrets of the PowerBuilder Masters

<http://www.PowerBuilderBooks.com>

EDITORIAL OFFICES

Phone: 914 735-7300

Fax: 914 735-3922

ADVERTISING & SALES OFFICE

Phone: 914 735-0300

Fax: 914 735-7302

CUSTOMER SERVICE

Phone: 914 735-1900

Fax: 914 735-3922

DESIGN & PRODUCTION

Phone: 914 735-7300

Fax: 914 735-6547

WORLDWIDE DISTRIBUTION by
Curtis Circulation Company

739 River Road, New Milford, NJ 07646-3048

Phone: 201 634-7400

DISTRIBUTED in the USA by
International Periodical Distributors

674 Via De La Valle, Suite 204

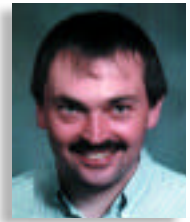
Solana Beach, CA 92075

Phone: 619 481-5928

**SYS-CON
PUBLICATIONS**

GUEST EDITORIAL

Jim Redman



Java: The Smarter Choice?

Information systems, meaning primarily software, are increasingly seen as a competitive weapon by which faster development and deployment equals a business advantage. This means more features on a shorter deadline. For developers this can be achieved by working harder or, perhaps the smarter choice, adopting Java. While some organizations reject Java because it doesn't have the right brand name or they view it as a risk, a recent International Data Corporation study reports significant cost and time savings by organizations who use it.

Of course, I didn't really read the study because it confirms what Java developers already know: there are solid technical reasons why Java development should be faster and more robust. The developers of the language had the advantage of history and, for the most part, picked the best of other languages and skipped the worst. Java itself lacks many of the "hacks" present in other languages, which makes it easy to learn and use. The language has evolved rapidly and well, and although it may not be perfect, it's good and improving. The most fundamental property, ByteCode portability, makes building cross-platform applications easy - a huge savings since very few enterprises are totally homogeneous. The Internet ancestry and network support within Java are obvious advantages if you're building distributed systems. JDK and Swing provide a feature-rich class library. Graphical layout tools allow even those of us who are artistically challenged to create fairly good-looking user interfaces. Even the humble JavaDoc takes some of the pain out of the necessary task of documentation. Java 1.0 was viewed as a way to create bouncing images on a Web site, and Java 1.1 is a real enterprise business tool.

What's hard to find through statistics is that something about Java development just feels good. Unlike other languages, coding in Java rarely seems to reach a point where the language itself is the limiting factor. Part of the reason for this is that at runtime the objects can tell you about themselves. This capability is most obvious in the concept of an "interface," where the object is irrelevant and only the methods matter. This "dynamic runtime" allows you to ask objects what features they can provide while the application is running. While this may seem trivial, it leads to the powerful component architecture of Java: JavaBeans.

JavaBeans are usually manipulated in a graphical editor. While GUI builder capabilities are available in other languages, JavaBeans can tell the development environment much more about themselves than their simple design properties. Applications such as Sun's Java Studio, combining JavaBeans and other dynamic Java objects, allow nonprogrammers to simply

and elegantly create whole applications. In an eerie video-gamelike world, mouse click-and-drag maneuvers connect the on-screen dots and build complete and powerful systems. Don't try this in other languages. In the area of factory automation, organizations have spent millions of dollars largely to re-create, in less dynamic languages, the capability that is available in an \$89 software package, Java Studio.

Once you can ask a component about itself, you can begin to ask for not just an object, but for something that provides the functionality you need. This is a subtle but important distinction because it allows greater flexibility and removes the need to re-create software as options change. A tool to "output a document" may be a printer or a formatter and an e-mailer. In an ideal environment the application searches the network for the required functionality and uses it. Java's platform independence makes this possible. This could include not only hardware resources, but also sources of information and even filters to analyze that information. This is the essence and goal of Sun's JINI technology. It's also a logical extension of existing Java technology.

There are a great many other Java ideas and technologies either available or in progress. Many will succeed and become so commonplace they'll no longer appear exciting. In software, as in real life, imagination is limited by the language used, so we can expect new ideas and concepts both incremental and revolutionary that are today literally inconceivable. We're working with first-generation Java technology; JDK 1.2 has not yet even been released. The programmer's toolbox for JDK 2.0 (and 3.0, 4.0, etc.) will certainly contain items that have previously been impractical or haven't yet been imagined.

Java is changing the software development process. Building the components will be the task of the developer; building the enterprise will be the task of a systems engineer. You may view change as a risk, but risk is always part of an equation that includes reward, and the rewards are potentially large. There's little doubt that Java can provide a competitive advantage, and that the technology will advance in ways that traditional development won't be able to match. ●

About the Author

Jim Redman is the president of ErgoTech Systems, Inc., a company focused on developing Java applications and toolkits for plant-floor automation. This includes links to low-level systems and hardware, and also network links - including CORBA support - for enterprise distribution of factory automation information. He may be reached at JRedman@ergotech.com.

Collections

The new collections API framework will change the shape of Java's current data structures

by Bhaven Shah

As Java matures, new sets of behavior added to its API allow developers and programmers to write more sophisticated programs with less difficulty. This article focuses on collections API, a new abstract data structure that will be a part of the core Java Development Kit (JDK) 1.2 API. (The collection classes are also available as an add-on package for the JDK 1.1 class libraries.)

Overview

An object-oriented application often consists of classes that need to refer to a collection of values of a given type. JavaSoft supports this capability by means of its rich set of collection classes – abstract data types for modeling “collections” of objects. Collections are a recent addition to the JDK. A collections API is a unified framework for representing and manipulating collections, allowing them to be manipulated independently of the details of their representation. Some examples of collections include:

- Set: a mathematical set of elements with no duplication
- List: an ordered sequence of elements
- Map: an array of key-value pairs

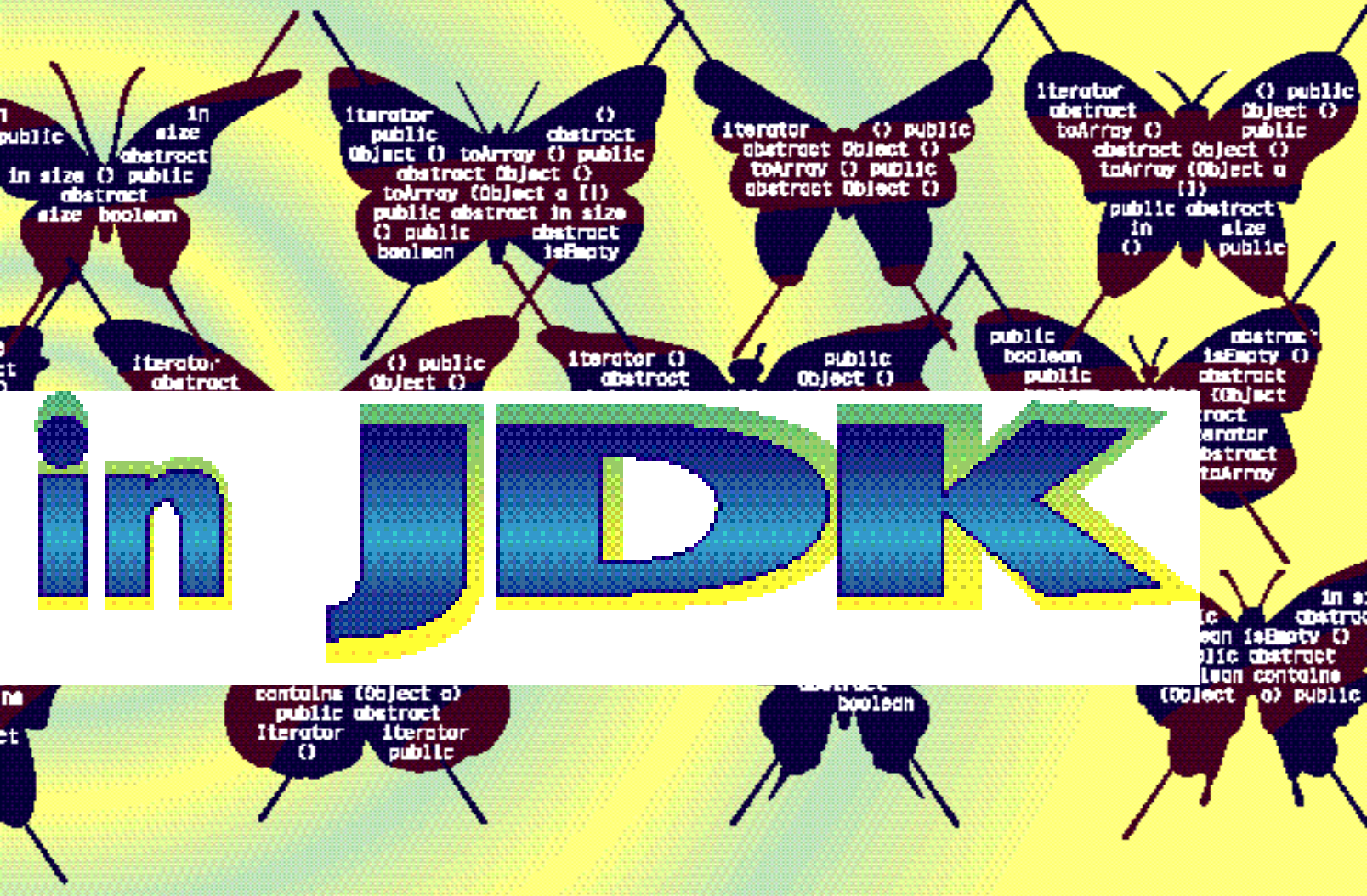
- Vector: an array of elements that can grow or shrink dynamically

Collections can be used to represent relationships among objects in a data model. These relationships may be one to one, one to many or many to many, depending on the model. While a simple array can be used to represent a one-to-one relationship, collections of objects are more appropriate for representing one-to-many and many-to-many relationships. Since the JDK 1.2 is still in beta, this article refers to the collection package released for JDK 1.1 unless explicitly stated otherwise.

Collections Framework

The collections framework in Java is used to separate the abstract notion of a collection of data from its data structure via the use of interfaces. For example, consider a List interface versus a linked-list implementation. A linked list can implement a List interface that may be used by other classes independently of how it is implemented by the class that uses it. The advantage of specifying interfaces in this fashion is that you can manipulate the collections without having to know the details of how a specific collection is implemented. The primary advantages of collections framework in Java can be summarized as follows:

1. Interoperability between unrelated APIs: Two different applications with an underlying data structure implemented separately can exchange data using the collection interface API since they both conform to the standard collections API.
2. Less effort in learning new APIs: If two applications exchange a list of objects, the lists can manipulate each other even if they are populated with their application-specific data objects



- since they both conform to the same API for manipulating lists.
3. Easier design and implementation of APIs: Application designers using the generic API that relies on collections won't have to start the design of their applications from scratch every time.
 4. Software reuse: Programmers can implement generic, reusable data objects that conform to the standard collection interfaces. These objects can also implement algorithms that operate on collection objects independent of the details of their representation. These data objects and algorithms can be reused in different applications.

Java Collections vs JGL and ODMG Collections

The Java collection framework is not the first abstract data class library for Java. Java Generic Library (JGL) and Object Database Management Group (ODMG) collections offer similar functionality. As we focus on the advantages of the JDK collections, it might be worthwhile to compare them with the proprietary but powerful and widely used JGL and ODMG's collection interfaces. The main design goal of JGL, an already existing collection package from ObjectSpace, was consistency with the C++ Standard Template Library (STL). It has approximately 130 classes and interfaces.

The JDK collections framework, on the other hand, is meant to be simple and lightweight. The whole collections package contains about 25 classes and interfaces. So, for developers who are used to the sophisticated JGL library, switching over to JDK collections might not be very attractive. However, as the Java libraries mature, the JDK collections library is also expected to grow. In addition, the collection classes are a part of the JDK and

hence do not have to be installed separately.

The JDK collection interfaces are similar to the ODMG's specifications (spec. 2.0) of Java interfaces released in May '97. The JavaSoft collection interfaces serve as the root of an interface inheritance hierarchy and the ODMG interfaces are derived from them. Again, the significant advantage of using the JDK collections is portability, as the applications using the JDK collections won't have to use any other classes or libraries outside the core JDK classes.

Collection Interfaces and Classes

The JDK Collections API consists of four core collection interfaces that represent different types of collections (such as sets, lists and maps):

1. **Core interfaces**
2. **Concrete implementations** (implementations for the core collection interfaces)
3. **Abstract implementations** (partial implementations of core collection interfaces to facilitate custom implementations)
4. Some **basic algorithms** such as sorting arrays, lists, searching arrays

Core Collection Interfaces

There are four core interfaces in the Java collections hierarchy. The collection interface is the root in the collection hierarchy. Two of the interfaces (Set and List) are children of the collection interface and add more methods to it; the last interface, Map, represents a mapping of keys to values. Each interface is described in detail below.

Core JDK interfaces

<|> indicates an Interface



New JDK collection interface hierarchy

Interface java.util.collections.Collection

```
public interface Collection
public abstract int size()
public abstract boolean isEmpty()
public abstract boolean contains(Object o)
public abstract Iterator iterator()
public abstract Object[] toArray()
public abstract Object[] toArray(Object a[])
public abstract boolean add(Object o)
public abstract boolean remove(Object o)
public abstract boolean containsAll(Collection c)
public abstract boolean addAll(Collection c)
public abstract boolean removeAll(Collection c)
public abstract boolean retainAll(Collection c)
public abstract void clear()
public abstract boolean equals(Object o)
```

java.util.collections.Collection class

Interface java.util.collections.Set

```
public interface Set extends Collection
public abstract int size()
public abstract boolean isEmpty()
public abstract boolean contains(Object o)
public abstract Iterator iterator()
public abstract Object[] toArray()
public abstract boolean add(Object o)
public abstract boolean remove(Object o)
public abstract containsAll(Collection c)
public abstract boolean addAll(Collection c)
public abstract boolean removeAll(Collection c)
public abstract boolean retainAll(Collection c)
public abstract void clear()
public abstract boolean equals(Object o)
public abstract int hashCode()
```

java.util.collections.Set class

Interface java.util.collections.List

```
public interface List extends Collection
public abstract int size()
public abstract boolean isEmpty()
public abstract boolean contains(Object o)
public abstract Iterator iterator()
public abstract boolean add(Object o)
public abstract boolean remove(Object o)
public abstract containsAll(Collection c)
public abstract boolean addAll(Collection c)
public abstract boolean removeAll(Collection c)
public abstract boolean retainAll(Collection c)
public abstract void clear()
public abstract boolean equals(Object o)
public abstract int hashCode()
public abstract Object get(int index)
public abstract Object set(int index, Object element)
public abstract void add(int index, Object element)
public abstract Object remove(int index)
public abstract int indexOf(Object o)
public abstract int indexOf(Object o, int index)
public abstract int lastIndexOf(Object o)
public abstract int lastIndexOf(Object o, int index)
public abstract void removeRange(int fromIndex, int toIndex)
public abstract boolean addAll(int index, Collection c)
public abstract List(Iterator listIterator)
public abstract List(Iterator listIterator, int index)
```

java.util.collections.List class

Collection

The collection interface represents a group of objects that may or may not be ordered. The base collection is duplicate-free and mutable. This class provides the base interface that most of the interfaces in the java.util.collections package extend.

Set

A set is a collection in which no duplicate elements are permitted. Ordering (if any) is generally established at Set creation time.

List

If you want to use a collection that can hold duplicate objects and have specific ordering, you need to use this interface (also known as a Sequence). With it, the caller generally has precise control over the position of each element in the list.

Map

This interface represents a mapping from keys to values. Each key can map to, at most, one value. This interface doesn't extend the collection interface.

Concrete Implementations

Classes that provide concrete implementations for the collection interfaces can be used either directly or subclassed to add additional behavior to the collection interfaces. The important classes in this category are:

1. **HashSet:** A set of classes backed by a hash table, this serves as a good, general-purpose Set implementation.
2. **TreeSet:** A balanced binary tree implementation of the SortedSet interface; unlike HashSet, a TreeSet imposes ordering on its elements.
3. **ArrayList:** A resizable-array implementation of the List interface (essentially an unsynchronized Vector), this class was provided in addition to Vector for consistency in naming and behavior.
4. **LinkedList:** A doubly linked List; this class may provide better performance than ArrayList if elements are frequently inserted or deleted within the List. It's useful for queues and double-ended queues (deques).
5. **Vector:** The new Vector class is a synchronized, resizable-array implementation of a List with additional "legacy methods" from the existing JDK1.1's java.util.Vector class. Changed from the legacy Vector class in JDK, the java.util.Vector class in JDK1.1 currently extends

java.lang.Object class. The new collections-based Vector class extends java.util.collections.AbstractList class (which in turn extends from java.util.collections.AbstractCollections class) and implements java.util.collections.List interface.

6. **HashMap:** A hash table implementation of the Map interface (essentially an unsynchronized Hashtable), the class is provided in addition to the Hashtable, for consistency in naming and behavior.
7. **TreeMap:** A balanced binary tree implementation of the SortedMap interface; unlike HashMap, it imposes ordering on its elements.
8. **Hashtable:** A synchronized hash table implementation of the Map interface with additional "legacy" methods from the existing JDK1.1 java.util.Hashtable class. Changed from the existing Hashtable class in JDK, the Hashtable class currently extends java.util.Dictionary class. The new Hashtable class implements Map interface in addition to extending the Dictionary class. Also, in the new Hashtable implementation any non-null object can be used as a key or as a value, unlike the previous implementation of Hashtable.

Anonymous Implementations

The new collections framework provides java.util.collections.Collections class, which has implementation methods accessed solely through public static factory methods. This allows any arbitrary algorithmic operations on collections.

Views

The collections framework provides users with the flexibility to work with either the actual collections (where they can perform read-write operations) or just a view of the collection (where they can perform only read operations), depending on the application needs. For example, the java.util.collections.Collections has methods like unmodifiableCollection (Collection c), which returns a view of the collection c backed by the original collection that was passed in. Other similar methods are unmodifiableCollection, unmodifiableSet, unmodifiableList, unmodifiableMap and unmodifiableSortedSet. All these methods return an unmodifiable view of a specified collection that throws an UnsupportedOperationException if the user attempts to modify.

Abstract Implementations

The classes in this category provide a skeletal implementation for the core collection interfaces to minimize the effort required to implement them. The JavaDocs

Ad

Interface java.util.collections Map

```
public interface Map
public abstract int size()
public abstract boolean isEmpty()
public abstract boolean containsKey(Object key)
public abstract boolean containsValue(Object value)
public abstract Object get(Object key)
public abstract Object put(Object key, Object value)
public abstract Object remove(Object key)
public abstract void putAll(Map m)
public abstract void clear()
public abstract Set keySet()
public abstract Collection values()
public abstract Set entries()
public abstract boolean equals(Object o)
public abstract int hashCode()
```

java.util.collections.Map class

New JDK Vector class hierarchy

Public Interface Map



AbstractList extends AbstractCollections and implements

New JDK Hashtable class hierarchy

<|> indicates an Interface



new Hashtable hierarchy

Collections class

```
public class Collections extends Object
public Collections()
public static void addToList(List l)
public static void addToList(List l, Comparator c)
public static int binarySearch(List l, Object key)
public static int binarySearch(List l, Object key, Comparator c)
public static Object min(Collection c)
public static Object min(Collection c, Comparator comp)
public static Object max(Collection c)
public static Object max(Collection c, Comparator comp)
public static List subList(List l, int fromIndex, int toIndex)
public static Collection unmodifiableCollection(Collection c)
public static Set unmodifiableSet(Set s)
public static SortedSet unmodifiableSortedSet(SortedSet s)
public static List unmodifiableList(List l)
public static Map unmodifiableMap(Map m)
public static SortedMap unmodifiableSortedMap(SortedMap m)
public static Collection synchronizedCollection(Collection c)
public static Set synchronizedSet(Set s)
public static SortedSet synchronizedSortedSet(SortedSet s)
public static List synchronizedList(List l)
public static Map synchronizedMap(Map m)
public static SortedMap synchronizedSortedMap(SortedMap m)
public static List newList(int n, Object o)
public static Enumeration enumeration(Collection c)
```

java.util.collections.Collections class

API documentation for these classes describes precisely how each method is implemented so the implementers will know which methods should be overridden, given the performance of the "basic operations" of a specific implementation. Following are some of the key abstract collection classes:

1. **AbstractCollection:** This class provides a skeletal implementation of the collection interface. The implementation of this class is neither a Set nor a List, but a general-purpose collection.
2. **AbstractSet and AbstractMap:** This class provides a skeletal implementation of Set and Map interfaces.
3. **AbstractList:** This class provides a partial implementation of a List backed by a random-access DataStore (such as an array).
4. **Comparator:** This class represents an order relation that may be used to order a collection or sort an array. Comparator may be used instead of a Comparable type's natural ordering or to order elements of a type that does not implement a Comparable interface.
5. **SortedSet:** This class represents a Set whose elements are automatically sorted, either in their natural ordering or by a Comparator provided at SortedSet creation time.
6. **SortedMap:** This class represents a Map whose mappings are automatically sorted by key, either in the key's natural ordering or by a Comparator provided at SortedMap creation time.

Algorithms

The new JDK collections package provides algorithms that operate on collections, their views and wrappers. The algorithms are provided in the java.util.collections.Collections class, which has several public static methods that operate on collections or return collections. Some of the important algorithms are described below:

1. **Collections.sort(List):** Sorts a List using a merge sort algorithm that provides average-case performance comparable to a high-quality quicksort, guaranteed $O(n \cdot \log n)$ performance (unlike quicksort) and stability (unlike quicksort). (A stable sort is one that does not reorder equal elements.)
2. **Collections.binarySearch(List):** Searches for an element in an ordered List using the binary search algorithm.

3. **Collections.min(Collection) / Collections.max(Collection):** Returns the minimum/maximum element in a Collection.
4. **Iterators:** In addition to the Vector and Hashtable changes, one of the main changes in the JDK data structures is the collection framework's new Iterator interface, which is useful for iterating over the Collection objects. The Iterator differs from the existing Enumeration class in JDK with respect to its additional power to modify the collection while iterating over it. For example, you can remove elements from the backed collection while iterating over it. In addition, Iteration is extended to provide ListIterator interface that provides easier bidirectional iteration over lists. Enumeration will remain a part of the java.util package, mainly for the backward compatibility with the legacy JDK API.

The following code snippet demonstrates the use of Iterators. Vector v has already been created and populated with data objects:

```
Iterator i = v.iterator();
while (i.hasNext())
{
    Integer intObj =
(Integer)i.next();
    System.out.println("Integer value:
" + intObj.intValue());
    // remove this element from the
collection using Iterator
    i.remove();
}
```

Another main feature of the collection framework is the synchronized collection wrappers: java.util.collections.Collections class provides methods to obtain synchronized wrappers backed by standard (typically unsynchronized) collection. This allows programs to carry out thread-safe (synchronized) operations on the collection objects.

The new JDK collection implementations are unsynchronized (with the exceptions of the Hashtable and Vector classes), but may be synchronized externally with synchronizing wrappers. All have fail-fast iterators that throw a runtime exception in response to concurrent modification of the backing collection rather than behave non-deterministically.

Sample Example Using Collections API

Listing 1 indicates how collection classes can be used to represent and manipulate data in a simple but efficient manner. This example demonstrates some of the features of the new collection-based Vector class

Ad

AbstractCollection class

```
public class AbstractCollection implements Collection
public abstractCollection()
public abstract Iterator iterator()
public abstract int size()

public boolean isEmpty()
public boolean contains(Object o)
public Object[] toArray()
public Object[] toArray(Object a[])
public boolean add(Object o)
public boolean remove(Object o)
public boolean containsAll(Collection c)
public boolean addAll(Collection c)
public boolean removeAll(Collection c)
public boolean retainAll(Collection c)
public void clear()
public void toString()
```

java.util.collections.AbstractCollection class

Interface Iterator

```
public interface Iterator

public abstract boolean hasNext()
public abstract Object next()
public abstract void remove()
```

java.util.Collections.Iterator class

and Comparator interface. The example creates a Vector and fills it up with integers. It then obtains an array representation of these elements by calling toArray() method, performs a sorting operation using the Comparator interface and, at the end, demonstrates how to use the Iterator to retrieve each element of the vector and how to remove elements from the collection using the Iterator.

Here is the output obtained by running the example code in Listing 1:

```
$ java Demo
```

Given Vector before sorting:

```
20, 500, 400, 100, 300,
```

Sorted vector after modifying the array:

```
20, 100, 300, 400, 500,
```

Vector using Iterator:

```
20, this element has been removed from the
vector
100, this element has been removed from the
vector
300, 400, 500,
$
```

Using Collections with JDK 1.1.x and 1.2

With JDK 1.1.x

The collections framework API has been

released as an add-on for the preexisting JDK 1.1 API. The collection classes in the add-on package are direct copies of JDK 1.2 counterparts and differ only in their package names. The APIs are identical with one significant exception: as they are in different packages, they are distinct Java types and cannot be cast/assigned between each other.

If required, however, source code using the 1.1 APIs can be easily (mechanically) ported to use the JDK 1.2 counterparts. The 1.1 collection classes packaged under com.sun.java.util.collections are currently available as a separate download from the core JDK 1.1 API at JavaSoft's Web site.

With JDK 1.2

In JDK 1.2 the collections API framework will be part of the core Java API and packaged under java.util package. This package is available today with beta releases of JDK 1.2. This means that if you use the collection package with JDK 1.1.x, you'll be importing collection classes using import statements similar to the one used in Listing 1. When you move your application to JDK 1.2 later on, all you do is change these import statements to import java.util classes.

Providing Implementation for Collection Interfaces

If you're implementing any of the collection interfaces, keep in mind that all general-purpose collection implementation classes should provide two "standard" constructors:

- A void (no arguments) constructor, which will create an empty collection
- A constructor with a single argument of type Collection, which will create a new collection with the same elements as its argument

The latter constructor allows users to copy any collection, producing an equivalent collection of the desired implementation type. For example, all general-purpose Map implementations should provide a void (no arguments) constructor and a constructor that takes a single argument of type Map.

Migrating to the New Collections API Using JDK 1.1

What's the impact of changing your existing Java data structures to the new collection-based data structures? Let's see what's required to migrate your application so as to use the new collection interfaces and classes for JDK 1.1. The steps are given below:

1. Download the new Collections.zip for the JDK 1.1.x.
2. Add the collections.zip archive to your CLASSPATH.
3. Replace java.util Enumeration with com.sun.java.util.collections.Iterator.
4. Change package names of the classes you wish to change to the new collection-based classes (e.g., replace java.util.Vector with the new com.sun.java.util.collections.Vector) and change the corresponding method calls in these classes.
5. Applets or applications packaged as JAR files to be used in Web pages should reference or include the collections.zip archive in their packaging or CODEBASE. This will ensure that the implementation is available upon download to the browser environment.

Conclusion

Java's collections framework provides easier and more powerful use of its data structures than ever before. It greatly facilitates writing programs involving numerical calculations and algebraic operations, and as the framework matures we can expect solutions for more complex mathematical problems. Whether JavaSoft will provide any mechanism for efficiently passing collection objects using RMI in a distributed environment remains to be seen, but you can always count on JavaSoft! ●

Resources - URLs

1. JavaSoft: www.javasoft.com
2. JDK 1.1 download: Java.sun.com/products/jdk/1.1
3. Collections API download for JDK 1.1: java.sun.com/beans/infobus/index.html#COLLECTIONS
4. JDK 1.2 download: www.javasoft.com/products/jdk/1.2/index.html
5. ODMG: www.odmg.org
6. JGL: www.objectspace.com

▼▼▼▼▼ CODE LISTING ▼▼▼▼▼
The complete code listing for this article can be located at www.JavaDevelopersJournal.com

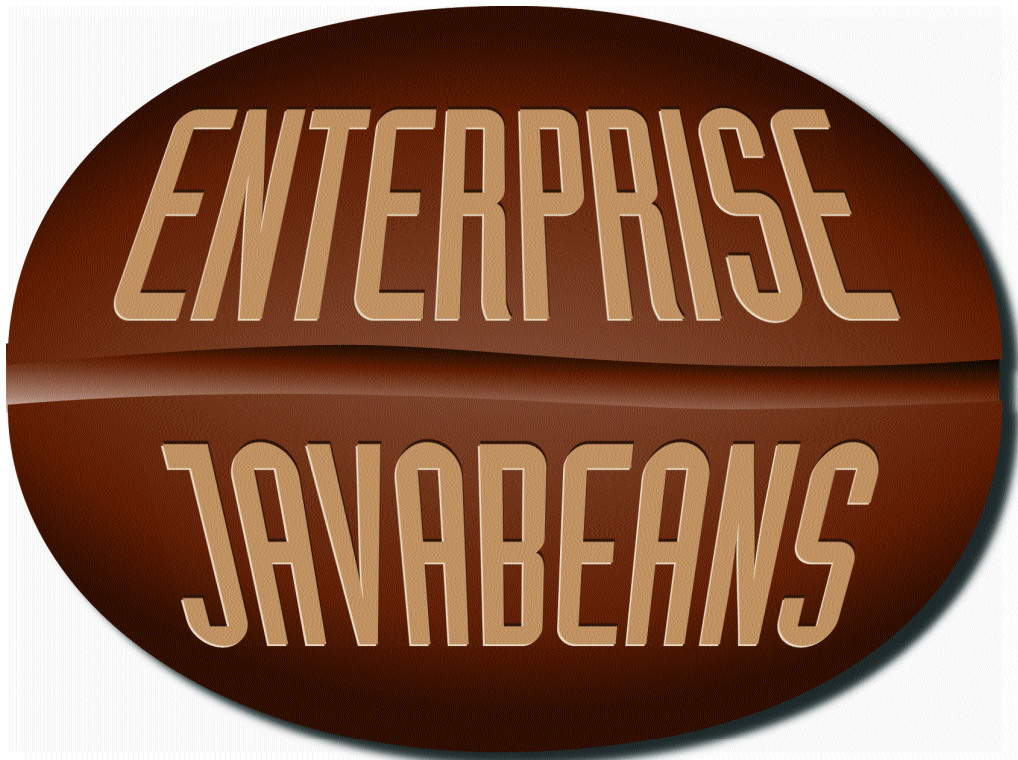
About the Author

Bhaven Shah, a member of the technical staff at i2 Technologies in Dallas, Texas, has a BS and an MS in computer science. His four years of programming experience include two years of Java and CORBA development. His focus is on GUI, client/server and distributed software development. Bhaven can be reached at bshah@i2.com.



bshah@i2.com

Ad



Industrial Strength

Increasing portability between servers after enterprise application has been developed

by Li ane Acker

Developing and maintaining distributed business applications is hard. As if writing business logic weren't hard enough, enterprise application developers have also been saddled traditionally with the daunting task of implementing transaction management, persistent state management, thread safety, resource pooling, security, distribution and life cycle/location of business objects. This makes enterprise application development more time-consuming and costly, and requires a broad range of expertise from the development team.

Once an enterprise application has been developed, using its code across other business applications becomes difficult because: (1) business logic is mixed with code to manage transactions, persistence, security and resources; and (2) other applications may employ different programming models. Ideally, one would like to reuse business logic as much as possible to ensure consistency of business rules across a company's applications.

Likewise, it's difficult to reuse code across platforms, servers

and backing stores (e.g., databases) when the business application contains low-level API calls that are specific to a particular platform or product. Portability across platforms is particularly desirable when you want to support client access from customer systems. Portability across server and database products allows an application to be migrated to higher-end systems as the business grows and its needs change.

Enterprise JavaBeans is a new specification from Sun Microsystems that aims to simplify the development, maintenance and code reuse of multitier enterprise business applications. Enterprise JavaBeans (EJB) is, foremost, a component model; that is, it facilitates reuse of software building blocks (components). A component model specifies the way a component interacts with its environment and other components, and offers programmers guidelines for building components so they can be dynamically composed with other components.

Enterprise JavaBeans is an object-oriented component model that focuses on the development and packaging of multitier enterprise Java applications. Like JavaBeans components, Enterprise JavaBeans are objects that can be used without source code because they can be customized through their external properties. Unlike JavaBeans components, Enterprise JavaBeans are always accessed remotely, as part of a distributed, multitier application. Such an application is typically characterized by a thin client (one containing only presentation logic) accessing remote business objects. The business logic and data access logic reside on one or more servers. The benefits of multitier applications include scalability, performance, reliability and flexibility; such applications are well suited to high-volume business transactions and Web-based business applications. Because the client contains only presentation logic, multitier applications can support a wide variety of client devices.

Goals of the Enterprise JavaBeans Specification

A major goal of the EJB specification – to simplify the programming model for distributed enterprise applications – is accomplished in four ways:

1. EJB is based on the Java programming language.
2. It's also based on the JavaBeans component model, an easy-to-use programming model that relies on simple programming and naming conventions for creating reusable, portable, customizable Java components. Enterprise JavaBeans are special, nonvisual JavaBeans that run on a server.
3. EJB includes a set of high-level APIs for transaction, security and persistence management; these APIs must be supported by an EJB-compliant server product.
4. The EJB programming model allows the programmer to focus on the business logic rather than coding thread safety, concurrency, resource pooling, security checking and transaction management. The EJB-compliant server product is required to perform these services automatically on behalf of the Enterprise JavaBean.

A second major goal is to enable reuse. EJB inherently enables reuse across applications because it's based on an object-oriented programming language (Java) and a component model (JavaBeans), and it inherits the cross-platform portability of Java. EJB goes further, however, enabling reuse across servers, transaction managers, database products and application-assembly tools. It does this by standardizing – and requiring servers to automate – services like transactions, security and persistence.

Each Enterprise JavaBean is required to implement certain interfaces so as to allow the server to manage it, and the server is required to give the Enterprise JavaBean control at certain well-defined execution points. This programming model allows the programmer to isolate the business logic, yielding greater portability.

A third major goal of the EJB specification is to support heavy-duty business applications – those that are distributed, scalable, transactional, multiuser, secure, persistent, flexible, high-performance and CORBA-interoperable. CORBA is an industry-standard architecture published by the OMG consortium for distributed, cross-language object servers. Because EJB is CORBA-interoperable, Enterprise JavaBeans can be hosted in CORBA servers and

Each container hosts one or more EJB-Home objects that manage a single class of Enterprise JavaBeans. An EJBHome provides a remote interface that client applications can use to create new (or find existing) Enterprise JavaBean instances of the class that it manages. An EJBHome also provides a naming context for the Enterprise JavaBeans that it manages. This means that client applications can use the Java Naming and Directory Interface (JNDI) to look up the EJBHome in the namespace – without a priori knowledge of where in the network its server resides – to obtain a stub to the EJBHome.

For each Enterprise JavaBean instance residing in the server, there is a corresponding object called an EJBObject, which represents the client view of the Enterprise

“The creation and finder methods can take arbitrary parameters, except that one finder method is required to take an instance of the bean's primary key class as input, and they must both return an instance of the bean's

accessed by non-Java clients or by ActiveX clients using a COM-CORBA bridge. CORBA interoperability will be achieved by enhancing Java RMI to communicate using a CORBA-compliant wire protocol called IIOP. This enhanced implementation of RMI is typically called “RMI over IIOP.”

Enterprise JavaBeans Architecture

Figure 1 illustrates the processes (shown as boxes) and objects (shown as ovals) that exist at runtime in an Enterprise JavaBeans application. The client application process contains only presentation logic and stubs to remote business logic objects. As in Java RMI, the client code interacts with the stubs as if they were local objects, and the stubs present the same interface to the client as the remote business objects.

The EJB server process hosts one or more EJB containers. An EJB container provides the application context for Enterprise JavaBeans, management and control services, and manages security, distributed transactions and state persistence for them. It is expected that each EJB-compliant server product will provide at least one EJB container.

JavaBean. It provides a remote interface consisting only of the Enterprise JavaBean's business methods, which a client application invokes via a stub to the EJBObject. The EJBObject's role is to intercept all client requests (even those coming from other EJBs residing in the same server) so that qualities of service for transactions, security and so forth can be maintained transparently to both the EJB and its client. When the EJBObject receives a business method invocation, it delegates to the business logic implemented in the Enterprise JavaBean.

Of all the runtime objects shown in Figure 1, only the Enterprise JavaBean is implemented by the developer. The remaining object implementations are generated by the server/container product's EJB deployment tools. This gives the developer a simple programming model, one that allows the business logic to be isolated and hence portable across EJB server products.

Session Beans vs Entity Beans

The Enterprise JavaBeans specification gives the developer flexibility by supporting multiple types of components. One distinc-

tion made in the specification is between session beans – transient, nonrecoverable, unshared components that represent operations to be performed on behalf of a client – and entity beans – persistent, recoverable, shared components that typically represent data backed by a database or other backing store, identified by a unique primary key.

A session bean exists (logically) for the duration of a single client/server session dedicated to one client. Although these beans can access and update a persistent store, they typically do not, and the container provides no support for session bean persistence other than notifying the bean of transactional boundaries. Session beans are typically either stateless service provider components or components that maintain conversational states with a particular client. The container is required to manage the conversational state of a session bean, saving it if the bean is passivated by the

mapping between the persistent store (e.g., a relational database table's columns) and the bean's container-managed public fields. By contrast, a bean-managed entity bean performs all of its own persistent data management. This might be useful when the bean accesses legacy data in a backing store not supported by the server/container product on which the bean is deployed.

In addition to persistent storage management, EJB-compliant server/container products are also responsible for providing services such as life-cycle management, concurrency, distribution, exception handling, transactions and security.

Writing an Enterprise JavaBean

To write an Enterprise JavaBean, the developer must write two Java interfaces and one Java class. The two interfaces provide the client view of the Enterprise JavaBean's EJBObject and EJBHome; the class

javax.ejb.EJBObject) includes all – and only – the business methods that the bean exposes to client applications. For example, a HotelRoom EJB might provide a method to reserve the hotel room it represents for a specified number of days. The bean's home interface (extending javax.ejb.EJBHome) provides methods of two kinds: creation methods (which must be named "create") and finder methods (which must begin with "find"). Finder methods are allowed only on the home interfaces of entity beans, not session beans. The creation and finder methods can take arbitrary parameters, except that one finder method (named findByPrimaryKey) is required to take an instance of the bean's primary key class as input. The creation and finder methods must return an instance of the bean's remote interface (or finder methods can return a collection of instances).

Listing 2 shows the bean implementation that one might write, corresponding to the interfaces shown in Listing 1. The bean class must implement either the javax.ejb.EntityBean or the javax.ejb.SessionBean interface. This example shows an entity bean; the details of writing a session bean vary slightly.

The bean implementation contains four kinds of items. First, the bean implements all the business methods from the corresponding EJBObject interface. Second, the bean implements methods from the EntityBean (or SessionBean) interface. These methods exist primarily to allow the bean to get callbacks at certain execution points; the bean developer is not required to do anything in these methods. The exception is that when developing a bean-managed entity bean, the bean developer must perform the appropriate backing-store synchronization in the ejbLoad, ejbStore and ejbRemove methods.

Third, for each create method in the bean's home interface, the bean must implement a corresponding ejbCreate and ejbPostCreate method, and its parameters should match the create method. For a container-managed entity bean, the ejbCreate method returns void, and is responsible for initializing the bean instance's container-managed public fields based on the input parameters. (The container is responsible for later updating the backing store based on the instance's field values.) For a bean-managed entity bean, the ejbCreate method returns a primary key value computed from the input parameters, and is responsible for initializing the bean's fields and updating the backing store. The ejbPostCreate method for both kinds of entity beans returns void; it allows the bean developer to perform postcreation computation, but often it would do nothing.

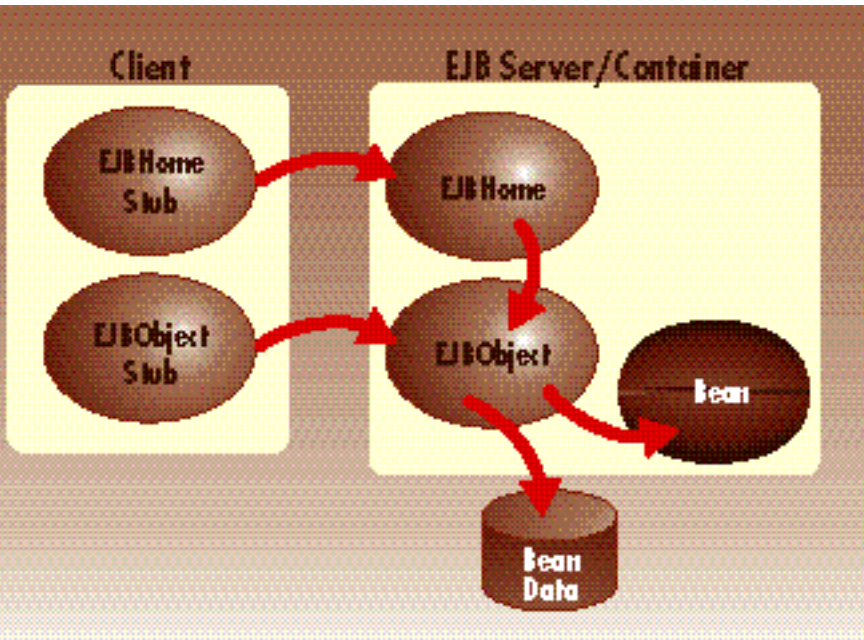


Figure 1: Processes and objects that make up an Enterprise JavaBeans application

server and restoring it when the bean is reactivated.

Another distinction made in the specification is between container-managed and bean-managed entity beans. The distinction refers to how the persistent state of the EJB is managed. The former implementation contains no code to open, access or update a backing store; the container is responsible for managing the persistent store on behalf of the bean. This would include, for instance, reading a row in a relational database in order to initialize the bean's container-managed fields and updating the database before passivating the bean. When deploying a container-managed entity bean to run in a particular server/container product, one would use the deployment tools provided by the product to describe the

provides the business logic (the bean implementation). Listing 1 shows two interfaces one might write for an entity bean representing a hotel room record in a database. The bean's remote interface, HotelRoom, must extend the EJBObject interface from the java.ejb package (a package introduced by the EJB specification). The bean's home interface, HotelRoomHome, must extend the javax.ejb.EJBHome interface. Both javax.ejb.EJBObject and javax.ejb.EJBHome extend the java.rmi.Remote interface, meaning that these interfaces can be invoked remotely via a stub. Because they are Remote interfaces, every method must include the java.rmi.RemoteException in its throws clause, as well as other application-specific and EJB-specific exceptions.

The bean's remote interface (extending

Ad

For bean-managed entity beans only, for each finder method in the bean's home interface, the bean must implement a corresponding `ejbFind` method, with parameters that match the home's finder method. The `ejbFind` method must compute and return a primary key value based on the input parameters.

Fourth, a container-managed entity bean exposes its container-managed data as public fields. The container is responsible for reading and writing these fields as it updates and accesses the backing store.

Notice that the EJB implementation need not contain any code to access a backing store, manage transactions, impose security constraints or ensure thread safety. The services provided by the server/container allow the developer to focus almost entirely on the application's business logic.

The Enterprise JavaBean developer finally packages the two interfaces and one class that constitutes the EJB into a special kind of zip file called an `ejb-jar`. The `ejb-jar` also contains other classes on which the bean depends: a manifest file that describes the `ejb-jar`'s contents and a serialized deployment descriptor object. The deployment descriptor indicates what runtime services the bean requires from the server/container and how the container should manage the bean, particularly with respect to life cycle, persistence, transactions and security. The environment properties embedded in the deployment descriptor allow an application developer to customize the bean for a specific use. For instance, environment properties might indicate the name of a database to be used by a bean-managed entity bean, or indicate the JNDI names of other EJBs that it uses.

Writing an Enterprise JavaBeans Client Application

Listing 3 shows pseudocode that illustrates how one would write client code to remotely access an Enterprise JavaBean and its home. (The requisite exception-

handling code is omitted for brevity.) The client application programming model illustrated would also apply when one Enterprise JavaBean is using another Enterprise JavaBean.

First, the client application must locate the bean's home object, using JNDI. To do so, the client application must have the bean's JNDI name, which it could obtain dynamically from a property setting. The client initializes JNDI by creating a `javax.naming.InitialContext` object, performs a JNDI lookup, passing the bean's JNDI name (e.g., "applications/hotel/rooms"), then narrows the return value to an instance of the bean's home interface. (Think of a narrow operation as a typecast that works on remote objects.) The result is an RMI stub to the bean's home object, residing in a server somewhere on the network.

Second, the client application uses the bean's home to create a new bean instance or find an existing bean instance in the server where the home object resides. For entity beans, creation results in adding new data to the persistent store; finding simply activates (if necessary) a new instance of the bean class to represent data that already exists in the persistent store. Hence, client applications typically use a home's create methods when working with session beans and a home's finder methods when working with entity beans. The result of invoking a creation or finder method on the home stub is another RMI stub, an RMI stub to the bean's `EJBObject`.

Third, the client application invokes business methods on the bean's `EJBObject` via the stub returned by the home. This is done exactly as if the bean implementation were being used as a local object, except that the client application must always be prepared to catch a `java.rmi.RemoteException`.

When the client application finishes using the bean, it will sometimes want to remove it. For an entity bean, removing the bean results in removing data from the persistent store.

As this is typically not desired, the `remove` method is usually invoked only when using session beans. When a client application is finished using an entity bean whose persistent data should remain in the backing store, the client application need not perform any action to release the bean other than allowing the local stub object to be garbage-collected. (This is true of the home object also.) It is the server's responsibility to eventually passivate the object in the server and allow the bean to be garbage-collected.

Conclusion

The Enterprise JavaBeans specification has received broad industry support. Major vendors, such as IBM, Oracle, Sybase and Netscape, have participated in the specification, and many more vendors plan to support EJB in their server and transaction-monitoring products. Because EJB was designed to be compatible with existing products, support for EJB should roll out rapidly. Several companies demonstrated early EJB support prototypes at the JavaOne conference in March, just weeks after the EJB 1.0 specification was published. One such company was IBM, which announced that they plan to support EJB across their software middleware and application servers, including Component Broker, TXSeries, CICS/390, MQSeries, DB/2, IMS and Lotus Domino, and that EJB is a key element of IBM's Network Computing Framework. ☛

About the Author

Liane Acker develops Enterprise JavaBeans server runtime and tool technology for IBM. She has worked in the areas of object-oriented and distributed programming for six years, with particular focus on CORBA, JavaBeans and Enterprise JavaBeans. In 1992 she received her Ph.D. in computer science from the University of Texas at Austin. Liane can be reached at lacker@us.ibm.com.



lacker@us.ibm.com

◆ LISTING 1.

```
public interface HotelRoom
    extends javax.ejb.EJBObject {

    public void reserve (int numberOfDays)
        throws java.rmi.RemoteException;

    // <more business methods only>
}

public interface HotelRoomHome
    extends javax.ejb.EJBHome {

    public HotelRoom create (RoomIdentifier roomNo)
        throws java.rmi.RemoteException,

    javax.ejb.CreateException;

    public HotelRoom findByPrimaryKey
        (RoomIdentifier roomNo)
        throws java.rmi.RemoteException,
        javax.ejb.FinderException;

    // <more create or find* methods only>
}
```

▼▼▼▼▼ CODE LISTING ▼▼▼▼▼

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

Ad

JWizard



Everything should be as simple as possible, but no simpler—Albert Einstein

by Claude Duguay

User interface design can be a real struggle when one of the requirements is to make programs accessible to a larger market. The most suitable metaphor for a given domain may not be simple enough for inexperienced users, yet the program still has to address their needs. If you can explicitly lead users through each step and get them to their objective, their net experience is usually positive. This month we'll build a framework you can use to develop your own sophisticated Java wizards.

Basic Design

Figure 1 shows a screen shot of the JWizard class in action. The WizardImage panel on the left is responsible for displaying an image, border and the basic spacing. The buttons and beveled line above them are part of the WizardNavigator panel. The rest of the display changes as you move forward or backward, and is contained by a panel that uses the DeckLayout (a layout manager that works like the familiar CardLayout, but overcomes fundamental focus management problems by disabling components when they are made invisible).

The JWizard framework design lets you drop components, usually JPanel objects, in order and then manage the sequence more explicitly. It uses a WizardSequenceManager, which implements the SequenceManager interface. Since Wizards typically collect some kind of data, we implement a DataCollectionModel interface to store results (our default being a PropertyDataModel derived from Properties). Both the SequenceManager and DataCollectionModel can be replaced by other implementations if you choose.

We also implement a WizardValidator interface that lets you control whether the user can move forward or backward at any given point, supporting dynamic data validation on each panel. When changes are made to the DataCollectionModel, the JWizard framework is notified and checks

the current panel through this interface to decide whether it needs to update the button status. If the user can't move forward and/or backward, the button(s) are disabled.

The WizardPanel implements most of the page mechanics and was designed to be subclassed. It lets you create the larger underlined title and explanation text by default, though you can set one or both of these values to null for more control. The WizardPanel implements the WizardValidator interface and provides a few utility methods that make development easier.

Quick Tour

It may seem complicated to have a data model, sequence manager and validator interface, but the benefits are obvious when you start building with JWizard. Here's a quick example of how it works in practice. We collect some personal information and ask a simple question before producing a result panel. The logic for this simple application could become complicated if the framework didn't handle it well. The flow from one panel to the next is shown in Figure 2.

The first panel collects information and registers itself as a DocumentListener for each of the JTextField objects in order to update the data model as changes are made. Since InformationPanel is a subclass of Wiz-

ardPanel, we can call getDataModel() to get the model and the setValue method to update changes. The event handler calls the model's hasValue method to determine if a field has content. In this example, if all three fields have data, the canMoveForward flag is set to true. When the model changes, it notifies the framework, which then checks the WizardValidator interface to activate the button(s) as appropriate. The Next button becomes active only if all three fields have content.

The second panel sets up six JRadioButton objects, makes them part of the same button group and registers itself as an ActionListener for each button. When a button is pressed, the FavoritesPanel calls the WizardPanel superclass method getManager to get the SequenceManager, and then sets the next panel with the setNext method. At the same time, it sets the subsequent panel's next value to a null ("") string, which indicates a final panel in the sequence. When JWizard sees this, it replaces the Next button with a Finish button.

At the end of this sequence, if you press the Finish button, it will print out the PropertiesDataModel so you can see what you've collected. You can find all this code on the **JDJ** Web site (www.sys-con.com), along with the source for the entire JWizard framework. To try this example, just run the JWizardTest class.

Stacking the Deck

The DeckLayout and DeckPanel classes are fairly simple. The DeckLayout is similar to the CardLayout manager that comes with JDK, updated to eliminate deprecated calls and extended to handle focus traversal properly. Listing 1 presents the show method, which lets you select an active page, and the setActive method, which is shared by all page-switching calls. The setActive method enables or disables components and their children, and also makes them visible or invisible depending on the Boolean argument. The show method looks for and disables the active page before activating a new page.

Listing 2 shows a couple of calls from the DeckPanel class. In particular, the addPanel method automati-



Figure 1: Favorite Language

Ad

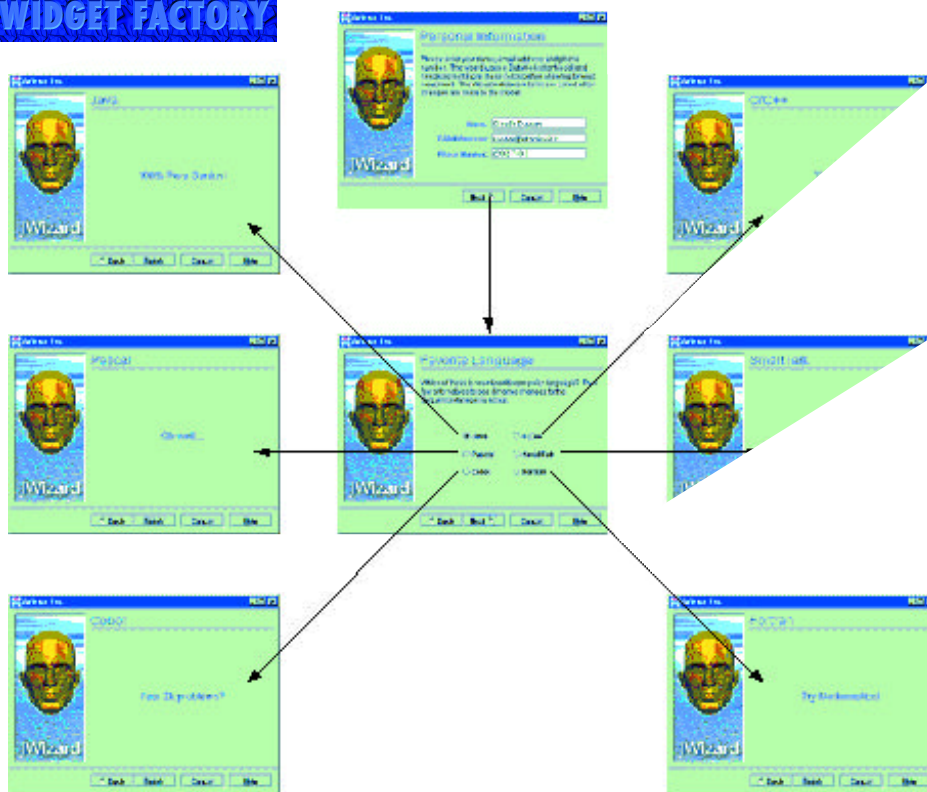


Figure 2: JWizardSequence

cally sets the sequence manager to the order in which panels are added. The first panel is set as the first active panel in the sequence; all others are set as subsequent to the previous panel. This simplifies the most common case and allows changes to be made to the sequence manager at runtime. The DeckPanel keeps references to the DeckLayout and SequenceManager, which are both used in the setPanel method. Only DeckPanel accesses the DeckLayout manager directly.

The SequenceManager interface is presented in Listing 3. Page names are used to keep things simple. The concrete implementation keeps track of the first and current positions, and holds two Hashtables that associate a given key with the next and previous pages, respectively. Listing 4 shows the member variables, as well as the getNext and setNext methods, for the WizardSequenceManager. Before setting a sequence pair, we always remove any previous links. The JWizard class lets you retrieve and set the sequence manager with the getManager and setManager methods.

Validated Modeling

Wizards typically collect some kind of information and then act on it. To make it as easy as possible to collect various kinds of information, we use a replaceable model called the DataCollectionModel. Listing 5 shows the interface for this model, which has methods to set, get and remove values, test for the existence of a field, and allow you to register and unregister a change listener. The listener is

automatically registered by the JWizard class when the setModel method is called.

The PropertyDataModel is the default implementation of our DataCollectionModel interface. It extends the Java Properties class, and simply stores and retrieves relevant values in a Properties object. Listing 6 shows how the addChangeListener method adds a listener to the listeners Vector. The fireChangeEvent method dispatches a ChangeEvent to registered listeners. Notice that we clone the list before iterating through the listeners so we can avoid concurrency problems. The setValue method puts the value into the Properties set and fires the change event. These events are fired only when the model changes, so the set and remove methods are the only ones that trigger it.

Listing 7 shows the WizardValidator interface, which contains only two methods: canMoveForward and canMoveBackward. These are verified for the active page whenever JWizard is notified that a change took place in the model. Based on the response, the Next button may be active or inactive, and the Back button may disappear.

Paneling Wizardry

The WizardPanel class extends JPanel and implements the WizardValidator interface (see Listing 8 for WizardPanel's source code). The constructor lets you specify a title and/or description for your page. These are both optional since setting them to null ignores them. The title is an underlined text label at the top of the page, and the description is a

word-wrapped text description that you might use to explain what the user is expected to do on the page. These elements are so common that it makes sense to keep this functionality in the superclass. They use the north part of a BorderLayout and leave you free to use the rest. Normally, you would add a panel to the center of the page and place your user interface elements on that panel.

The WizardPanel uses a pair of member variables to keep track of the canMoveForward and canMoveBackward flags for the WizardValidator interface. As a subclass, you can change these directly at your leisure. Also provided are a pair of utility methods for easily accessing the DataCollectionModel and SequenceManager. The getDataModel and getSequenceManager calls get and effectively casts the required information from the JWizard (parent) container.

In Practice

To put the JWizard class to use, you need to place it in a Window or Dialog box. Since JWizard is an extension to JPanel, you can embed wizards into any interface, not necessarily in a separate window. Listing 9 shows the source code for JWizardTest, which extends JFrame and sets the size to match a typical Microsoft Wizard; you can resize it if you like. We then create a JWizard instance with JWizard.gif as the left-hand image, and add the various WizardPanel (extension) pages. Finally, we set the first panel and add the JWizard reference to the center of the Frame. The main method creates a JWizardTest object and displays it on the screen.

The JWizard widget is a powerful tool for developing wizards under Swing. It demonstrates the use of several reusable interfaces and a flexible design. While it may not do everything you could possibly want, you're free to extend it if needed. The foundation is strong enough to apply in a production environment. I hope it serves you well. Next month we'll take a look at a JComponentTree widget that lets you create connected component trees with various alignment and orientation choices. ●

▼▼▼▼▼ CODE LISTING ▼▼▼▼▼

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

About the Author

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

 claudio@atrieva.com

Ad

ADVERTISER INDEX

Advertiser	Page	Advertiser	Page	Advertiser	Page
Borland www.borland.com	19 408 431-1000	KL Group Inc. www.klg.com	B/C 800 663-4723	Silverstream www.silverstream.com	83 888 823-9700
Bristol Technology www.bristol.com	75 203 438-6969	Live Software info@livesoftware.com	41 619 643-1919	Sockem Software www.sockem.com	65 814 696-3715
Coriolis www.coriolis.com	77 800 410-0192	Net Dynamics www.netdynamics.com	79 650 462-7600	Stingray Software Inc. www.stingsoft.com	2 800 924-4223
Greenbrier & Russel www.gr.com/java	25 800 453-0347	ObjectShare www.objectshare.com	43 800 973-4777	SunTest www.suntest.com	11 415 336-2005
Halcyon www.halcyonsoft.com	35 888 333-8820	Object Matter www.objectmatter.com	50 305 718-9101	Sybex Books www.sybex.com	63 510 523-8233
IBM www.ibm.com	58&59 800 426-5900	ObjectSpace www.objectspace.com	4 972 726-4100	The Object People www.objectpeople.com	23 919 852-2200
IEC-EXPO www.iec-expo.com	73 888 222-8734	Object Management Group www.omg.org	53 508 820-4300	SYS-CON Publications www.sys-con.com	71 800 513-7111
ILOG www.ilog.com	17 415 688-0200	Progress/Cohn & Godly www.apptivity.com	21 800 477-6473	Thought, Inc. www.thought.com	48 415 836-9199
Installshield www.installshield.com	13 800 374-4353	ProtoView www.protoview.com	3 609 655-5000	Visionary Solutions, Inc. www.visolu.com	50 215 342-7185
Inno Val www.innoval.com	38 914 835-3838	Roguewave www.roguewave.com	15 800-487-3217	WebMethod www.wbmethods.com	33 888 831-0808
Keo Group www.keo.com	22&37 978 463-5900	Sales Vision www.salesvision.com	47 704 567-9111	Zero G. Software www.zerog.com	6 415 512-7771

1/4 Ad

1/4 Ad



The Land of the Rising Sun

Lessons in patience

by Alan Williamson

Here we are again, back for another look at the underbelly of Java. Those of you that don't know what I write about, stay tuned; those that do, feel free to jump to the next paragraph. Straight talking is what we do here. We strip away all the hype and look under the cover of the Java engine to see what's really ticking. What you'll find here is something you won't read in any book or discover in any other column. I'm not out to win friends or butter up any company. I'm here to tell the truth, and I hope to get you, the developer, thinking and talking about Java.

Each month we address a potential problem and contrast it with a character attribute. In previous columns we looked at faith and trust. This month let's go for patience. Patience is one of those things we all have. Some of us think we have a lot of it until the moment we actually need it. For example, there really should be a law against teaching your loved ones how to drive. This takes all the patience in the world. It's been said that the world would be a much better place to live in if we all exercised a little patience.

People aren't the only ones to suffer from a lack of patience; companies are equally guilty. Pick up any magazine and you'll notice thousands of examples. The software industry is particularly bad with patience. It's amazing that consumer confidence is as high as it is considering the turnaround time and life cycles of some of the products. Some say it's progress; I say it's annoying. As soon as a product hits the shelves, a new version or updated feature is announced. Companies aren't allowing the market to absorb the product; they're exhibiting no patience, always rushing to be first to market. Let me explain.

N-ARY has recently returned from Tokyo. We had the good fortune to exhibit our company's services and products at JavaExpo98. It was the first time we'd been to Japan and we fell in love instantly. It was amazing to discover a city as big as Tokyo so clean and friendly. An advanced city as well. We all know how well the Japanese are at con-

sumer electronics. How many of us have a Sony, Panasonic, JVC or some other Japanese derivative in our home somewhere? They have this knack of taking something and making it better – and, ultimately, smaller. We assumed we'd find the same level of experience and expertise in their software. A misplaced assumption, we'd soon discover.

We arrived preshow day and – once we persuaded the security guard that we were the representatives from N-ARY – proceeded to locate our wee corner booth. We walked past Oracle's stand. As usual, a huge impressive stage was being erected that resembled something from a Michael Jackson concert, not a computer show. I noticed something rather strange. Of all the great things Oracle is doing with Java, what do you think got top billing on their stand? The JDBC driver! Whoopee!

I thought this ironic, considering the topic of last month's column (if you missed that one, then it's worth getting a back issue for the column alone!) and my grievance with Oracle. I continued to look at the sign. Excuse me? The JDBC driver? You come all the way to Japan to promote the JDBC driver? Now I'm confused. We proceeded to set up our stand to promote our servlets and the power Java can offer on the server side. We were also promoting our new server monitoring product, n-formant, which is entirely servlet-based. In the excitement of setting up, we never got a chance to look any farther than the two or three stands beside us.

Show day came, as did the general public. Being a Java expo, we expected the majority of people coming in would be familiar with Java. So we prepared our pitch on servlets and n-formant, and how the coupling of Java at the server side could produce an excellent solution. Warning bells started ringing midway through the first day when we realized we hadn't done that much servlet talking but had spooled oodles on Java. Being of a suspicious nature, I looked around to make sure we had set up in the right hall and hadn't accidentally had a stand in the neighboring

accountancy trade show. An easy mistake.

I know, I thought, let's go and see the Sun stand. That will put my mind at ease. I hadn't been to the Sun stand at this point and was very excited about seeing it (yes, I know, I must get out more!). The first thing I noticed was the big stage, stockpiled with Sun servers. Interestingly, they seemed to place a lot of emphasis on their hardware. Then I found the Java section. Here I went back in time.

The world of Java is moving fast, with lots of new, exciting APIs that allow the developer to do a host of things. Those of you who went to JavaOne this year know the sort of exciting demos Sun had to play with. Real eye-catching examples of where Java can be deployed. However, we saw none of this.

Nothing of worth was on the stand that hasn't been around for years. I kid you not. Not a single new API. The Java Web server wasn't even being demoed, let alone the servlet API. Other notable absentees included JavaPC, Media API, 3D/2D API and Beans. Would you believe that the bean concept was not being sold? I couldn't believe what I was seeing. I needed to speak to someone here.

Sometime later in the day the ISV manager, Tomohiro Yamazaki, came over and introduced himself to us. After we talked for a little I mentioned the lack of new Java technology, noting that even Oracle was only promoting JDBC. He wasn't surprised. He said that Japan as a whole was behind in the software market. We estimated around two to three years behind America. Java was experiencing a slow uptake in Japan, with the majority of developers still focusing on C++ and COBOL.

The reasons were many, ranging from the uptake of the Internet as a whole to the cultural differences in the way Japanese do business. Whatever the reasons, we saw similar feedback. I spoke to some of the Sun/Java server team in California when I got back, and they had experienced the same reaction when they gave a series of talks on servlets earlier in the year.

At first I was shocked and couldn't believe how behind they were; I felt exasperated at what seemed a very difficult sale. It was like selling car radios when you have to explain the merits of owning a car

in the first place. Not easy.

However, thinking about it all, the Japanese are doing something they are renowned for and we aren't: they are exercising patience, resisting the urge to jump on the bandwagon and follow the crowd. But in this case, is it a good thing?

My first reaction is no. Java is evolving with more and more features coming online all the time and the developers need to stay informed. If they don't, then old technology will be employed and they'll be left behind. But is this a bad thing?

I don't think so. The problem is that Java is evolving a bit too fast. How many of you have moved over to the new 1.1 API? And how many of you have had to redo all your applets to work undeprecatingly in the latest browsers? A large number, I suspect. Well, can I let you in on a wee secret? We're about to change APIs again; be prepared for another learning curve, another rewrite and another version number. 1.2 is nearly upon us now, and I'm left thinking, Is this it now?

The time has come for us to say **stop**. Let's use what we have already and show the world what Java is really all about. Sometimes I feel like an Olympic runner (sadly, I don't look like one!) at the starting blocks; every time I think I hear the starter gun I'm

told to return, it was a false start. Not only is this getting on the nerves of developers, it's not good for consumer confidence.

We've approached many large client companies that have plowed serious money into CGI/Perl solutions where a Java servlet solution was more applicable. The reason they don't use Java? **It hasn't settled down yet...When it's in version 2 we'll look at it...Is it usable yet?** These are just some of the comments we're getting, and I'm sure they're familiar to many of you as well.

Somebody at Sun has to stand up and say **stop**. Enough. Quit bringing out new APIs and allow us to get used to the existing one. Java is supposed to be "write once, run anywhere." It's more like "write once, run on any 1.1.x JVM anywhere." We're back into the old version number game again, and if we're not careful it's going to get worse before it gets better.

Sun needs to exhibit the same patience as Japan. It needs to hold Java back for a number of years, and allow it time to gain a foothold in the industry. I'm hoping this is what they're aiming for with the new 1.2 release, but only time will tell. Even at the server side, we need to be careful with the version numbers. This isn't supposed to be how the game is played.

As developers we need to learn to walk

before we try running. We need to stop upgrading for the sake of upgrading. Although it's fashionable to always use the latest technology, it isn't always practical. Remember the phrase used when companies fail? They grew too big too fast. I fear this is in Java's future. It's growing too fast and isn't allowing the industry to catch up.

I'd like to thank Tomohiro Yamazaki from Sun (Japan) for talking to me in depth about this whole issue. His thoughts and insight into Japan were most welcome. We thoroughly enjoyed ourselves in Japan, and I would like to thank Mayuko Hayashi for her kind hospitality and for looking after us throughout the four days.

Patience, that's all we need to practice, and maybe we can make the world a happier place after all. ☪

About the Author

Alan Williamson is on the board of directors at N-ARY Ltd., a UK-based Java software company specializing in JDBC and Java servlets. He has recently completed his second book on Java servlets. His first book looks at using Java/JDBC/servlets to provide an efficient database solution. He can be reached at alan@n-ary.com.



alan@n-ary.com

1/2 Ad

Ad



MindQ Java

by MindQ Publishing, Inc.

Your personal Java trainer

by David Jung



Java is unquestionably one of the hottest development languages to learn at this moment. There are hundreds of books about Java, covering all different levels. You can spend thousands of dollars on this subject and still not get any closer to Java nirvana. While you could spend your money on countless books or attend a Java class at some training center or college, MindQ offers another excellent way to learn Java: Computer-Based Training (CBT). You're probably thinking that CBTs are boring and you'll never get anything out of them. In reality, that holds true whether you take a class or read tons of books on the subject. CBT simply offers an alternative. As elegant as the Java language is, it's very dry when you try to learn it. MindQ does an exceptional job of keeping your attention glued to the subject matter through its interactive user interface.

The Learning Process

The MindQ courseware is multimedia-enabled. It ranges from video segments of some of the creators of Java to audio descriptions to exercises that students can work on to reinforce what they've just learned. If you're hearing impaired or just don't want sound, there's an Audio Text Display window that displays what's being said.

Each CD takes between six and 12 hours to complete. Don't feel obligated to complete them in one sitting though. If you want to stop in the middle of a lesson, simply bookmark the area you're in and end the program. When you come back, just go back to your bookmark and continue where you left off. Also, it's designed in a hypertext format so you can click elements on the screen to get more information, just like navigating

through a Web site.

As you start each course, you see and hear an orientation of what the course is about, and you see where it resides within the entire curriculum. If you want, you can skip the orientation and go right into the lessons. Figure 1 illustrates the basis of the user interface. On the left-hand side are navigation buttons. The Audio Text Display window can be moved around the screen if it gets in your way.

The bottom of the screen holds the audio mute button, the pause/play button (an indicator that animates when the audio is playing), a slider bar to show how many screens you need for the given topic and a message bar. Each CD is broken into lessons with various topics. As you go through each lesson, graphic illustrations and animation emphasize the points. At the end of each topic there's a short quiz

MindQ Java

MindQ, Inc.

11490 Commerce Park Drive
Suite 400 Reston, VA 20191

Phone: 703 262-6600

Fax: 703 716-0237

e-mail: info@mindq.com

Web: www.mindq.com

Samples of course available for download

MindQ's Java Training CDs

Essential Java Training: \$795.00

Advanced Java Topics: \$995.00

Developer Training for Java: \$1,595.00

called "Reality Check." The questions are based on items discussed during the lesson. They are either multiple choice, fill in the blank or matching. Figure 2 is a sample of one of these quizzes.

At the end of each lesson you're given the option to write a program, in some cases several programs, based on a story

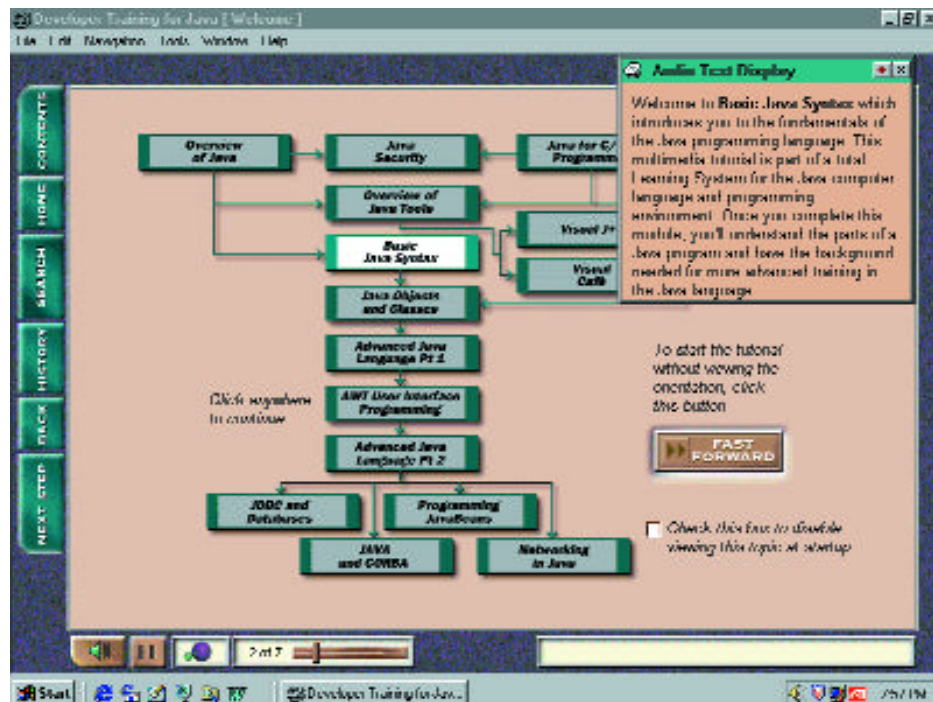


Figure 1

Ad

problem and subject that was covered in that particular lesson. The source code is made available to you so you can study it at your leisure.

The Curriculum

MindQ's Java courseware curriculum is laid out as if you were attending a Java course at a college or learning center. They offer three packages for you to choose from: Essential Java Training, Advanced Java Topics and Developer Training for Java. If you're new to Java or just want to learn the basics to prepare for the Sun Java Certification exam, the Essential Java Training course will prepare you. It covers all the basic subjects of Java development, such as:

- An overview of the Java Platform
- Basic Java Language Syntax
- Java for C/C++ Programmers
- Java Objects and Classes
- Advanced Java Languages
- AWT User Interface Programming

Advanced Java Topics takes you into the Java realm where Essential Java Training leaves off. This track will teach you the areas of Java that industry developers use to create and maintain client/server applications. Its subjects are:

- Using the Java Foundation Classes
- JDBC and Databases
- Programming JavaBeans
- Java Security
- Java and CORBA
- Networking in the Java Language

The Developer Training for Java is the complete library of Java courses, but it also includes a few more courses to round out the entire learning experience. It's broken down into four distinct areas of learning: Fundamentals, Intermediate, Advanced, and Tools for Java. The Fundamentals section covers the first four topics in the Essential Java Training course, including a course called "Java for Managers," which is less technical but offers a good understanding of the Java language, its benefits and how and when to use Java technology. Intermediate and Advanced tracks cover the remaining topics included in the Essential Java Training and Advanced Java Topics tracks. The Tools for Java contains three courses. One, an overview of Java technology tools, describes the various Java development environments on the market today with screen shots of each product. It discusses the features and benefits of each one in detail so you can make an educated decision when choosing which product to use. The other two courses cover Microsoft Visual J++ and Symantec Visual Café for

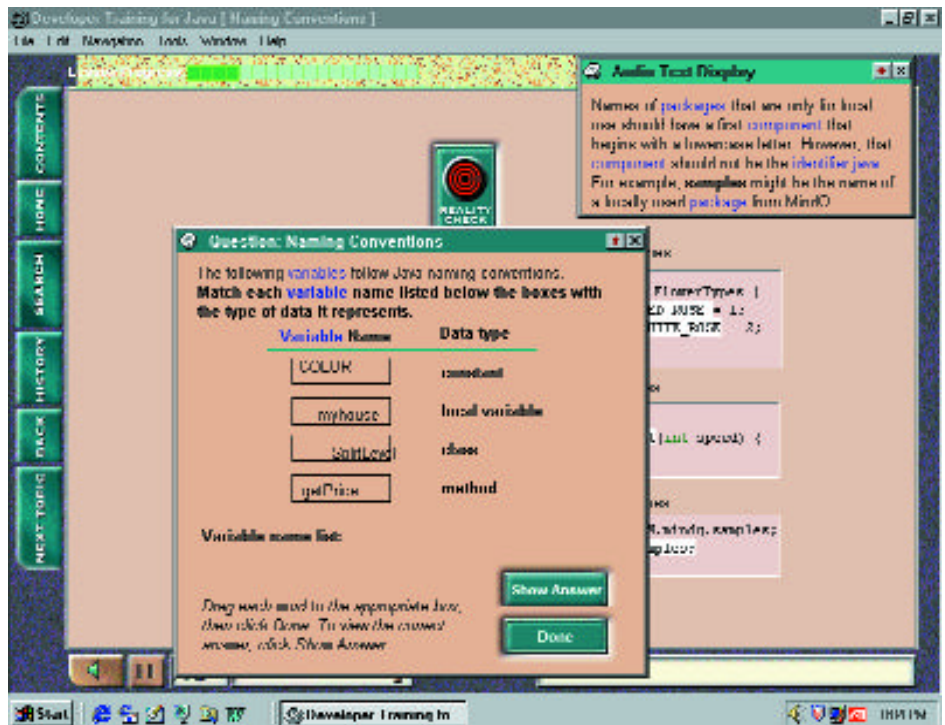


Figure 2

Java, probably the most popular Java development environments. They offer in-depth coverage on using the two products, maximizing their features to make Java applications.

Deployment Options

MindQ offers several methods for deploying these courses in your organization. You can run the CDs on a stand-alone workstation, distributed over your local area network or through your organization's intranet. In stand-alone mode you can run the course either straight from the CD or the local hard drive. To deploy the courses through your network or intranet, you'll need to obtain a corporate license, which includes all the courses plus a network installation package that allows you to choose the best deployment option for your organization. To run the courses over the network, your network administrator runs the installation package, which copies the courses to a server. The files are compressed on the server to save drive space. When a student wants to run a course, the client package on their workstation is started and the data streams to it as if the CDs were running locally. MindQ's streaming technology is what allows it to run over a LAN or an intranet.

Summary

I was pretty familiar with Java before reviewing these courses, but because I'm self-taught, some aspects of the language and environment weren't clear. I found

myself fascinated by the interface and MindQ's teaching method. From a learning standpoint the lessons are well organized. MindQ obviously gave their curriculum a lot of thought when they put the lessons together. The cost of each package reflects a one-year licensing term, but the CDs won't self-destruct at the end of the year. Think of it as a maintenance agreement. Any updates to the product are free of charge during the one-year licensing term. If you don't renew, you won't receive any updates.

After completing these courses, don't expect to suddenly be a Java programming genius. That will occur over time and after you've used Java for a while. You should, however, be able to sling around the Java buzzwords with confidence and have fairly intelligent conversations with other Java-savvy developers. If learning Java is what you're looking for, and you want to learn it at your own pace without spending a fortune on private lessons or wading through countless Java books, MindQ offers a great solution. ●

About the Author

David Jung works as a senior programmer analyst. He's a lead architect for all client/server development. He also coauthored several Visual Basic books, including Visual Basic 6 Client/Server How-To and Visual Basic 6 Interactive Course (Waite Group Press). He can be reached at davidj@vb2java.com.



davidj@vb2java.com



Java Studio

Sun Microsystems, Inc.

A visual authoring tool that uses JavaBean components to create Java applets, applications and new JavaBean components

by Dana Crenshaw



Java Studio comes loaded on a CD-ROM. Its packaging also includes an installation instruction booklet, a serial ard (for registration poses) and the manual,

Exploring Java Studio It tells you about the basics of Java Studio, and is good for getting started. It also contains added information on the examples, which are plentiful.

I recommend that you adhere to the minimum requirement for RAM. With 16 MB I was able to run the application; however, it was unbearably slow. An additional 16 MB would have made a noticeable impact. On one occasion I

opened up Paint in addition to working in Java Studio, and my system crashed. On another occasion I was attempting to close Java Studio and it got hung up on one of the save windows. I attribute both of these botches to the lack of available memory.

It took my Pentium 133 laptop approximately five minutes to install the application. It follows the typical Windows installation procedure. It is very basic; there are not a multitude of components to load with varying configurations (custom, typical, etc.). The system took approximately 35 MB of hard-drive space (with FAT32 I expect this total will be smaller).

As with most other applications, you can accept the suggested install subdirec-

Java Studio

Sun Microsystems, Inc.

901 San Antonio Road

Palo Alto, CA 94303

e-mail: webmaster@sun.com

Web: www.sun.com/studio

Test Environment:

NEC Versa 2435CD laptop

Pentium 133

16 MB RAM/1 GIG Hard drive/6X CD-ROM

Windows 95

System Requirements:

Sun's minimal requirements include a Pentium 100 MHz computer with 32 MB of RAM and a screen size of 800x600 min. The operating system can be Windows 95, Windows NT or a Solaris System (SPARC Platform Edition and Intel Platform Edition). Even though the 100 MHz is the minimum requirement, their recommended processor speed is 133.

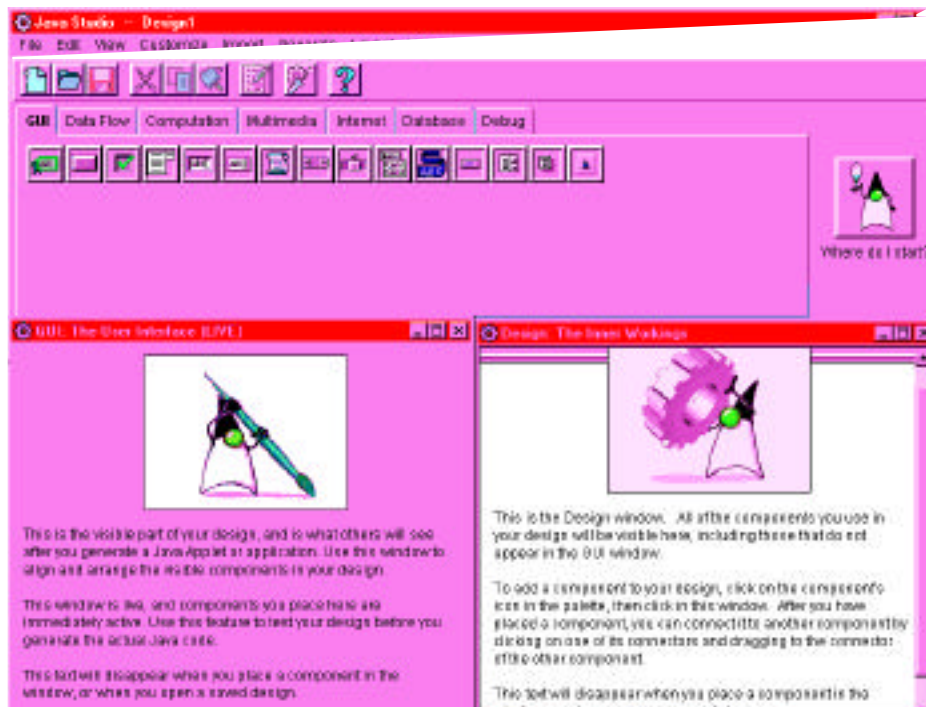


Figure 1: Java Studio interface

tory or enter another. If the subdirectory doesn't exist, you'll receive a prompt asking if it's okay to create one. Now, it does not attempt to install under the Windows Program subdirectory. This is not a really big issue, just something you should be aware of.

How It Works

Java Studio is a GUI tool for creating applets and applications by using JavaBeans. You can also create other JavaBeans for use in further development. The package includes JavaBeans that enable you to do the basics; however, you can add Beans that you have acquired to expand your creation capabilities. Unfortunately, it is not made clear how this happens.

There are three interfaces in Java Studio, as shown in Figure 1. The main window is the top window with the title "Java Studio - Design1." The tool bar gives you access to importing, saving files, generating (applets, applications or JavaBeans) and customizing the tab below, among other standard window functions.

The tab object below has tabpages for

Ad

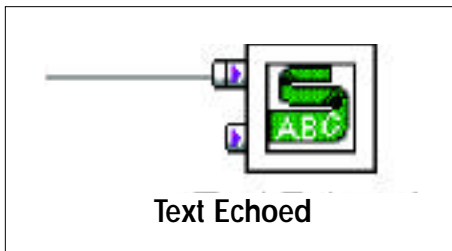


Figure 2: Ticker tape component

the different types of components at your disposal. The icons that appear below the tab headings represent the list of components for each type. For example, the selected tab in Figure 1 is GUI. The components listed here include Label, TextField, Text Area, ScrollBar and Ticker Tape. Under the Database tab you'll find icons for accessing a database and tables. Under the Multimedia tab you have icons for image map and sound player. These are only a few examples of the many icons representing available components. You also have the option of adding components from your library of JavaBeans.

The other two windows in Figure 1 are "GUI:The User Interface" and "Design: The Inner Workings." The GUI screen shows how your selected components appear in the applet or application. In this window you can move the components around by dragging and you can resize them as well,

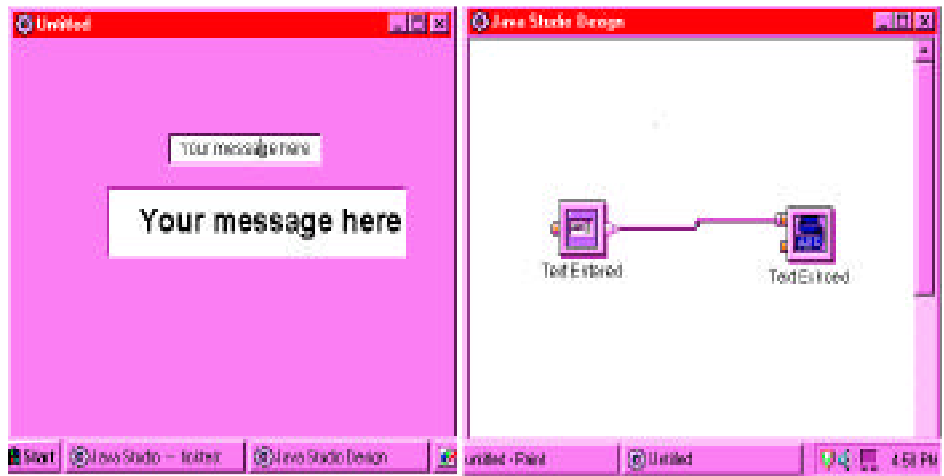


Figure 3: GUI and design window

so you get live feedback about the appearance and operation of what you're creating.

The Design window depicts the relationship between the components. These components have connectors that enable you to set the relationship for each component to the others. Figure 2 shows a ticker tape component. Notice the nodes on the left side.

The nodes are called **connectors**. The line coming out of the top connector actually originates from another component. It indicates that there is input coming into the Ticker Tape component from some other component. The Ticker Tape component has two input parameters (represented by the connectors). The connector with the line is for the text input, which is displayed in a scrolling fashion. The bottom connector is the input parameter for the scrolling speed. No input is required here; the default is set in what is called the **customizer**. The customizer can be thought of as the properties window.

When you put these two together, the text field and ticker tape, you get the result seen in Figure 3. You can see the line drawn between the output connector of the text field to the input connector of the Ticker Tape. That signifies that whatever is typed into the text field component is fed into the Ticker Tape, where it scrolls at the rate of speed indicated in the customizer.

Creating Things

The process of creating applets, applications and JavaBeans is a matter of selecting components, setting the properties in the customizer and then establishing the relationships among those components.

If you have complicated mathematical formulas, you'll certainly want to give thought to how this can be accomplished graphically within the framework of what Java Studio offers. In fact, with anything

you plan on creating you will want to give thought to its design. While you do have the luxury of seeing what is being created without having to compile and run, your logic can still go awry.

Recommendation

Java Studio is a good product that can be used by all levels of Java programmers. Beginners will need to work where they understand the components; otherwise this tool won't benefit them. While it can be used as a learning tool, without a basic understanding of Java components it will be one steep learning curve.

Advanced users may find the tool by itself to be limiting. However, with an expanding library of JavaBeans they should be able to overcome some of these limitations.

One thing I'd like to see improvement in is the manual. They should either expand **Exploring Java Studio** or include an additional manual to give more information on the components and how to create applets, applications and JavaBeans. I don't expect them to duplicate what is already in the marketplace in terms of Java programming, but I do expect them to give more than what they are currently providing.

Overall, Java Studio is a product worth considering if you're in the market for a Java tool. At less than \$100, it should be very attractive to both beginning- and intermediate-level developers. ●

About the Author

Dana Crenshaw is a software engineer and freelance writer. He currently works for TRW, Inc., and has over 15 years of experience in IT. He lives and works in the Atlanta area, and graduated from the Georgia Institute of Technology. You can contact him with questions or comments via email at danap@compuserve.com.



SYS-CON RADIO



Tune in for detailed discussion of products from JDJ advertisers!

Java readers voted #1 with their browsers!

2 million banners delivered each month BPA (more than all other Java media added together!)



CASE STUDY

by Cara O'Sullivan

Visual Café Smooths the Transition to Java



Visual Café employed in the migration of PeopleNet

It decreased the learning curve, was easy to debug and cut development time and effort in half

Xerox Uses Java Tool to Cut Development Time in Half

A global company in the document processing business, Xerox Corporation offers a wide array of products and consulting services including publishing systems, copiers, printers, scanners, fax machines and document management software along with related products and services.

Xerox started the office copying revolution with the introduction of its 914 copier in 1959. Today Xerox stands poised for the continued expansion of the global document processing market, already enormous at \$200 billion a year and growing 10% a year. Including Fuji Xerox, whose results are not consolidated in accounting statements, Xerox world-wide revenues in 1996 were \$25 billion, two thirds of which were generat-

ed outside the U.S.

In June 1996 Karen Mihara, development manager, and Jim Johnson, chief technologist at Xerox Corporation, faced a challenge. Their task: to migrate the Xerox PeopleNet, a client/server application that serves more than 33,000 U.S. employees, to the Web. Because Xerox PeopleNet is one of the company's most critical employee/management empowerment tools, the migration would have to be seamless. Employees use the application to access and update their personal information, reference policy and process manuals, review salary planning information, submit paperwork electronically and perform workflow processes.

"One of our goals in placing Xerox PeopleNet on our enterprise-wide intranet was to move to a more productive, object-oriented development environment," says Mihara. "We also wanted a Web-based application to ease deployment and support of Xerox's installed base of Windows, UNIX and Macintosh workstations."

But Mihara and Johnson didn't want to

have to start from scratch. They especially wanted to make sure they could still use the existing Oracle databases that contained the proven data sources and business logic for the client/server application. It was crucial that the new Web-based application be able to work with Oracle database technology.

Making the Evaluation

"It was also very important that we be

JClass Pure Java GUI Components

By Keith Schengili-Roberts

As Java is increasingly used in corporate computing environments, it's important that programmers have the proper tools to build robust applications quickly and easily. Often a lot of effort is spent building the graphical user interface (GUI), which enables the user to interact with a program. The problem is that many programmers feel they're reinventing the wheel when it comes to GUI programming due to existing usability guidelines incorporated into countless programs. Most programmers agree that they'd rather spend their time working on the core application rather than its GUI. But GUI has increasingly been recognized as crucial, and it makes the difference between usable and unusable programs. Designing and building the right GUI is therefore an important part of any application.

KL Group has been creating GUI components for developers for the past nine years, first with its series of XRT widgets for Motif developers, then with Olectra components for MS Windows developers and more recently with its line of JClass Java components. KL Group was the first to provide commercial-quality Java GUI components, beginning with

About the Author

Cara O'Sullivan writes about business and technology. You can reach her at caracomm@usa.net.

able to easily migrate our developers from Microsoft's Visual Basic, which we'd used for the client/server application, to Java for the Web-based application," says Johnson.

Adds Mihara, "What we needed was a Java graphical development environment that closely mirrored Visual Basic and gave us some of the same advantages. Visual Basic was quick and easy for our developers to use for prototyping and debugging. We needed that same functionality in a development environment for the Web."

Part of the challenge for Mihara and Johnson in evaluating Java development tools was that at the time Java was just emerging as an industry standard. They were concerned about finding a stable development environment that supported Java's still evolving features.

Mihara and Johnson looked at several Java development tools until they found a solution that fit their needs.

"Of all the Java development environments we looked at, Symantec's Visual Café for Java was the most stable and feature-rich," says Johnson. Symantec also consistently puts out Visual Café releases and updates in sync with Java's. For developers, this was critical.

Migrating from Visual Basic to Java-Based Visual Café

Visual Café made the transition to Java very smooth for the Xerox developers. Visual Café allows developers to use a visual development process similar to Visual Basic and other Integrated Development Environments (IDEs). User interfaces can be designed by dragging-and-dropping graphical components from a palette and setting properties from a property editor.

Visual Café also has an Interaction Wizard that will generate code to connect events and methods between selected components. Visual Basic does not generate

interaction between components.

In addition, Visual Café accommodates developers who don't want to use the Visual design (RAD) mode. Source code can be directly edited, compiled and debugged within Café.

Debugging

The features Johnson, Mihara and Xerox developers now depend on include a debugger and the software's ability to debug threads and third-party components.

Visual Café provides a comprehensive debugging environment that allows debugging of applets (in Java's Applet viewer, Netscape Navigator or Microsoft's Internet Explorer).

Says Johnson, "We have been able to debug third-party components, in-house-developed components, multithreaded applets, CORBA-based communications and multiple applets communicating via shared memory without any problems."

the release of JClass LiveTable in 1996. Since then the JClass family of JavaBeans has grown rapidly, and has been adopted and endorsed by leading IDE vendors including Inprise, Symantec, IBM, Sun Microsystems, Powersoft and SuperCede.

Since its inception at the very beginning of commercial Java development, JClass LiveTable has matured and is considered a

feature-rich product. The latest version, JClass LiveTable 3.5, is a highly flexible and powerful way for programmers to display and manipulate data in tables, and now boasts native data-binding capabilities. With JClass LiveTable developers can create forms, grids, tables and multicolumn lists; readily incorporate AWT, JClass BWT or Swing components in cells; and have full control over the user's ability to edit cell contents and traverse through the table. The product also comes with sorting and searching capabilities, and thanks to an innovative technique known as JCString, programmers also have total control over the colors, fonts, border styles and shading elements in a table.

One of JClass LiveTable's key features that makes it flexible and applicable in a variety of situations is its use of an MVC-style data architecture. LiveTable's data can

come from any object that implements the JClass LiveTable data interface, such as an array in memory, a file, a socket or a database via a JDBC ResultSet object. Data can be static or updated dynamically – JClass LiveTable's data architecture supports JavaBean-style events that can be fired from the data source to the table whenever changes to the data occur. While JClass LiveTable comes with a number of prebuilt data sources, its architecture gives developers the freedom to use LiveTable any way they want.

JClass LiveTable also gives developers flexibility in the way tables are displayed. Using a customizable renderer/editor architecture, the product allows developers to store the data in whatever format they want, and then display it either using one of the many cell renderers included or with custom cell drawing code. Cell editors can be customized the same way, allowing developers to, for example, display Boolean data as checkboxes, use combo boxes or completely customize the behavior of the table.

JClass LiveTable is also a 100% certified Pure Java component, which makes it ideal for use in Integrated Developer Environments (IDEs), and rounds out and expands the capabilities of the standard palette of JavaBean tools provided within most IDEs. The certification also provides custom developers with an objective assurance that the components run across all Java technology-enabled plat-

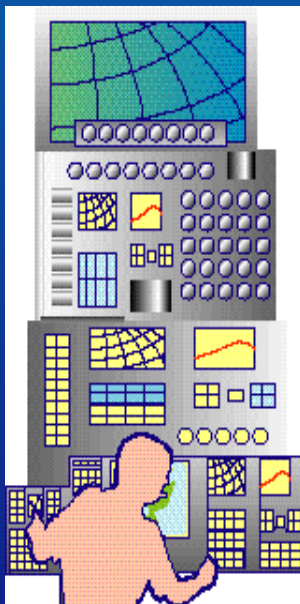
forms. KL Group is committed to delivering 100% Pure Java-certified components, and the JClass BWT (short for "Better Windowing Toolkit") was in fact the first commercial product to pass Sun's JavaBean certification process.

KL Group's family of Java components continues to grow with the recent release of JClass HiGrid, a database GUI, and JClass DataSource, a database JavaBean. JClass HiGrid enables developers to create, manage and update multilevel data-bound database components, while JClass DataSource is a powerful tool for data access, providing a common abstraction layer for multiple database connection techniques (such as a JDBC URL, a JBuilder DataSet or a Visual Café QueryNavigator instance). KL Group has also introduced two different suites of their popular JClass components for Java developers: a Standard Edition containing the basic set of JClass components and an enhanced Enterprise Edition designed specifically for building data-bound applications.

The family of JClass 100% certified Pure Java components and JProbe Profiler, the first in a series of planned, advanced diagnostic Java programming development tools, confirms KL Group's goal to help professional Java software developers be more productive and reduce the cost of building industrial strength Java applications. ☑

About the Author

Keith Schengill-Roberts is the Webmaster for KL Group and author of the forthcoming *Advanced HTML Companion* (second edition, Academic Press). You can e-mail him at ksr@klg.com.



Also, according to Mihara and Johnson, developers can now create their own components and then reuse them as standard objects across many different Java applets.

"In Visual Basic, if you wanted to reuse code between different programs, you had to copy the code over into the source code for each program," points out Mihara. "With this software you can set up components once, add them to the graphical tool palette and any developer can simply drag-and-drop the components into another applet.

"This also means that logic or components need only be written and tested once, which considerably shortens our testing cycle," Mihara continues.

For the Xerox PeopleNet Web project, developers relied heavily on components because of their inherent reusability and the ability to drive implementation and development standards. Visual Café has the very useful feature of being able to "import" custom components onto a development palette.

Mihara and Johnson's team developed two basic types of components: (1) a wrapper component of third-party or standard Java graphical or functional components,

and (2) custom graphical components.

The wrapper components were implemented to enforce look and feel, or standardize operational behavior by restricting properties or methods behaviors. A big benefit of these types of components included simplifying the development environment so junior developers could come quickly up to speed.

Examples of wrapper-type components were a table component (the base component was KL Group's LiveTable component and a CORBA server connection class). The connection class, which interfaces to an Oracle database via Persistence Software's PowerTier product, was implemented in such a way that the CORBA connection could be shared among the various applets a user would request during a session. The connection class was implemented through the use of a common class that utilized static variables. A simple method from that class, `setObjectServer()`, is found in Listing 1.

Functionally, the establishment of a method like `setObjectServer()` has several advantages. First, it allows a standard simple interface for developers to establish CORBA connections. Second, it saves on client and server resources as each applet can share a single CORBA connection and limit the times the CORBA connection is initiated.

The components developed for Xerox PeopleNet Web fell primarily into the user interface realm of the product. The lightweight components do not use operating system "peers," and therefore provide a common look across all platforms. This is important in this case because Xerox has a heterogeneous desktop environment and a very large user base to support (more than 36,000 users). Examples of these types of components include a standard applet header, text label, custom buttons and tab control.

Easing the Learning Curve

For Johnson, the object orientation of Visual Café decreased the learning curve for his developers. "You drag-and-drop the GUI component you want, use a wizard to connect the component's interactions to

LISTING 1.

```
static XPNServer.XPN ServHel per;
static Persistence.ObjectServer OSHel per;

public void setObjectServer() throws org.omg.CORBA.SystemException
{
    Common tc = new Common();
    hostName = tc.getServer();

    if ((OSHel per == null)) {
        try {
            ORB orb = org.omg.CORBA.ORB.init(myAppletInstance, null);
            IE.Iona.OrbixWeb.Features.Config.setConfigItem

("IT_BIND_USING_IIOP", "true");
            OSHel per = Persistence.ObjectServerHel per.bind(markerServer, host-
Name);
            ServHel per = XPNServer.XPNHel per.bind(markerServer, hostName);
            connection = null; //Reset Oracle DB connection

            .. etc ..

        } catch (org.omg.CORBA.SystemException se) { throw se; }
    }
}
```

other objects and the software generates all of the Java code. You can then look at the code that was generated and learn Java that way," Johnson says. "In fact, not one of our developers has any formal training with Visual Café or Java. They've been able to learn at a pace that is good for them as well as for our schedule. And we didn't have to hire any new employees or bring in new consultants."

Mihara said she likes what she hasn't heard about using the software from the developers. "With Visual Basic, the applications tended to be very difficult to debug. Its close ties to the Windows operating system made the applications behave differently, depending on the Windows version and the other applications that were installed. Things were unpredictable," she says. "But I haven't heard any of our developers voice these complaints about Java and Visual Café. When you run Java applets in Visual Café, they're consistent. It's a much more stable and consistent development environment."

Johnson calculates that using the tool has cut development time in half - or more. "Because we used Visual Café, it took dramatically less time for us to develop Xerox PeopleNet," he says.

"It's a good investment," he continues. "Just think about how much you pay your developers for their time to see what Java and Visual Café can save your company." ●

▼▼▼▼▼ CODE LISTING ▼▼▼▼▼
 The complete code listing for this article can be located at www.JavaDevelopersJournal.com

✉ caracomm@usa.net

SYS-CON RADIO

LIVE coverage
 Java Business Expo
 Java Developer's Journal
 Award Ceremony

Ad

Cold Fusion
1/2 Ad

PBDJ
1/4 Ad

6.0
1/8 Ad

CPD
1/8 Ad



Optimize it!

by Intuitive Systems

A powerful CPU and memory profiler helps make your Java programs run faster

by Achut Reddy



Every chance I get, I lobby for performance tools for software developers, because performance tuning is hard. This is especially true in modern object-oriented languages

like Java, as opposed to older languages like C where the programming model was much closer to the hardware model. Furthermore, performance can affect the user's perception of your software. Hence, for any serious software-development project it's critical to have good performance tools available to assist the tuning efforts. Optimizelt 2.0 Professional from Intuitive Systems is one such tool.

Optimizelt claims to work with most Java 1.1 VM's including the following:

- Sun Microsystems JDK 1.1.1 through 1.1.6
- Borland JBuilder 1.0
- Symantec Café 2.5

Test Environment

I tested Optimizelt on a Windows NT 4.0 system with a 200 MHz Pentium, 32 MB of memory (the minimum supported configuration) and the standard Sun JDK 1.1.6 installed. I ran Optimizelt on several Java programs including one 200,000-line application.

Installation and Documentation

The installation was extremely smooth and uncomplicated. It required a mere 8 MB of disk space for the install. No hard-copy documentation was provided; however, the online documentation included a user manual (36 pages when printed) and a short tutorial. Both are in HTML format and viewed through your Web browser.

Two Tools in One

Optimizelt actually contains two tools, a CPU time profiler and a memory profiler

integrated into one GUI. Often, time and memory profilers are offered separately, but either one by itself gives you only part of your program's performance. By including both in a single program, Optimizelt allows you to see both sides of the coin.

Using the Product

After specifying the program to run, such as CLASSPATH, source path(s) and then arguments, press the Start button in Optimizelt to run the target program. It can be paused at any time to study the performance data up to that point and then resumed later. By default the memory profiler is always enabled and collects memory data continuously as the program runs, although it can be explicitly disabled if you wish. While the memory profiler is collecting data, it can be viewed in real time on the memory profile screen.

For the CPU profiler you explicitly start and stop recording performance data. Generally, you would run the program up to the point where you want to begin measuring and press the Start CPU Profiler button. Then you wait until the program has finished the operation you want to measure and press the Stop CPU Profiler button. An option is provided to pause automatically before the start of the program in case you want to profile it from the very beginning. Unlike memory data, CPU data can't be viewed in real time as it's being collected. The CPU data becomes viewable when you stop recording.

Optimizelt is an interactive, live analysis tool. As long as the program and the JVM are live, you can get performance information. Once the program exits, however, the performance data is no longer available. Hence, an option is provided to disable exits so that even if the program calls System.exit(), the JVM is forced to remain alive so you can obtain the data.

Optimize it! Professional 2.0

Intuitive Systems

599 North Mathilda Avenue, Ste 19

Sunnyvale, CA 94086

Phone: 408 245-8540

Fax: 408 245-8541

Web: www.optimizeit.com

E-mail: info@intuisys.com

Requirements: Windows 95/NT or Solaris 2.4, 2.5, 2.6 with 32 MB of memory

Price: \$389

CPU Profiling Features

The CPU profiler uses statistical sampling to determine how much time is spent in different parts of your program. The sampling interval is 25 milliseconds by default, but can be set as low as 5 milliseconds. Timing results are presented in a sorted hierarchical tree format that shows the top-level methods (e.g., the main() method in an application), their cumulative time (time spent in those methods plus all of their descendants) and percentage of total time (see Figure 1). You can then click on a node to cause it to expand and show all of the methods it calls directly, as well as their associated times. In this way it's easy to "drill down" the dynamic call graph of the program until you get to a leaf method. There is an option to invert the tree so that the bottom-most leaf methods are shown first and you can "drill up" to the top-level methods.

This tree display is intuitive and easy to use. Some things, however, are difficult to see in this display. For example, it's not possible to see all the callers and callees simultaneously for a given method. In order to do this, you must switch back and forth between normal and inverted modes.

Also shown on the CPU profile screen is a "hotspot" display, which lists methods sorted by the total time spent in each one, regardless of who called it. This is the display that clearly tells you which methods to focus your optimization efforts on.

The really great feature about the CPU

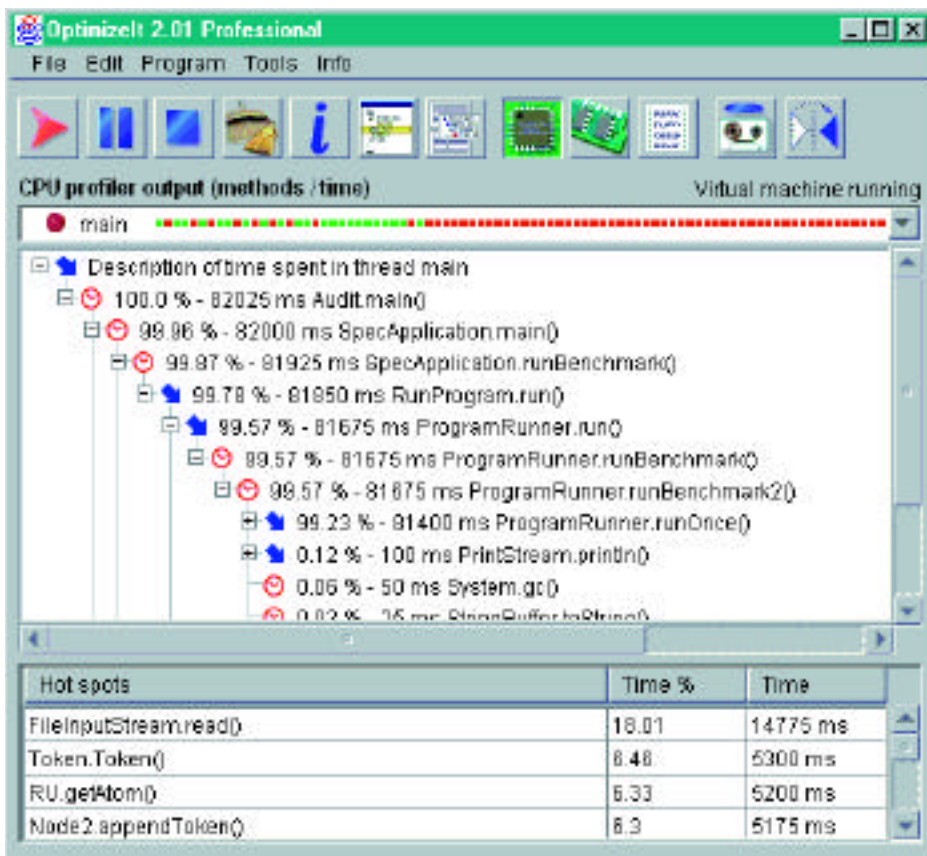


Figure 1: The CPU profiler shows cumulative method times in a hierarchical tree.

profiler is that data can be viewed for a particular thread or thread group; there's a hierarchical list in the program, including system threads (such as the AWT event thread) and a time line for each thread that shows when it was in a running or suspended state. This alone is a useful display, especially if your program has several threads. You can then select a thread or thread group from the list, and the tree and hotspot displays will show only data for that particular selection.

By default, the display shows real time (CPU plus wait time); however, you can opt to see CPU time only. This is useful for tuning algorithms as it relates directly to the number of instructions executed.

Memory Profiling Features

The memory profiler shows object allocations aggregated by type. As previously mentioned, this display is continuously updated as the program runs, showing the current instance counts (and, optionally, sizes) for every object type. It would have been nice if it computed totals for you as well. There's also a checkpoint feature that allows you to set a baseline for the instance counts. Subsequent measurements will then show the counts relative to the baseline. They'll go up as new objects are allocated, and go down again when the garbage collector frees them. There's an option to disable the garbage collector, which is useful if you

wish to see the total allocations over the life of the program.

Unfortunately, this display doesn't include arrays. Since arrays can account for a majority of heap space in many programs, I found this to be a significant omission, although Intuitive promises to include this feature in the next version. Another feature I'd love to see is the ability to show memory profiles by thread or thread group, just as the CPU profiler currently allows.

If you select an object type, you can then go to a second screen that shows, in a hierarchical tree format, the stack backtraces of all the places in the program where that type was allocated. You can go to a third screen that shows all instances of that type along with its values (the value shown is the result of calling the toString() method on the object). The instance screen also includes a unique and powerful heap analysis feature that shows you all incoming or outgoing references (but not both at the same time) to a given object instance. This is valuable because, although Java has an automatic garbage collector, it won't collect an object if you unknowingly keep a reference even though it's no longer needed. A heap analysis tool such as this is the only practical way to find such memory leaks. Apart from performance reasons, it can even be used as a debugging aid to inspect the connections

between your data structures and find possible errors.

Other Features

- A source viewer is provided to display source code whenever you click on a method name, provided you compiled the code with the -g flag.
- Printing is not supported directly, but you can export the data into HTML format and print from your browser. You can also export into ASCII or "importable" ASCII formats.
- Advanced users will appreciate that a set of API methods are provided to enable and disable the profilers at the program level so as to gain even finer control on the parts of the program to be measured.

Performance

Optimizelt generally performed well throughout my tests – a fact made more impressive considering it's written almost entirely in Java. Clearly the Optimizelt engineers must have used the tool on itself! I did find switching between screens a bit sluggish on my small memory (32 MB) machine, but presumably this would be less of a problem on a larger memory system. I ran a benchmark program to measure the amount of overhead profiling. For the CPU profiler by itself, a 1.2x slowdown; for the memory profiler, there was a 2.2x slow down. This is quite reasonable, and I think most users are willing to tolerate it in return for quality performance data. With both profilers running simultaneously, I measured a slowdown of 2.4x. While Optimizelt allows it, I don't recommend running both at the same time because the overhead of the memory profiler will affect the CPU profiler and can skew the results.

Summary

Optimizelt provides a CPU profiler and a memory profiler, both of which are needed as part of a full-performance tuning effort. It's easy to install and operate. The outstanding feature of the CPU profiler is its ability to partition the timing data by thread or thread group; for the memory profiler it's a heap-analysis capability that allows you to track all references to and from a given object instance. Optimizelt is a welcome addition to any Java programmer's arsenal of development tools. ☛

About the Author

Achut Reddy is a staff engineer at Sun Microsystems in the authoring and development tools group, which is currently working on Java performance issues. He can be reached via e-mail at achut.reddy@Eng.Sun.com.





Java APIs and Products for Consumer Devices

Providing a ubiquitous platform and language for communication

by Ajit Sagar

This month's discussion warrants a brisk walk down technological memory lane to examine Java's humble beginnings. In its original incarnation Java, then called Oak, was a language based on some of the features and syntax provided by C++. Oak was the result of Sun Microsystems's mission to find a way for consumer electronic appliances to communicate with each other. When Oak was launched, it didn't take off from the ground. Then came Mosaic, the Internet and the World Wide Web. Oak changed its name to Java, a marriage between Java and the Internet was made in enterprise heaven – and Java's exploits spread far and wide. Today Java's a language, a platform and an enterprise solution – a phenomenon.

But has Java abandoned its initial goals? Absolutely not. The language and the distributed enterprise computing solutions it provides are fast gaining market share in software for the enterprise. At the same time, Java has also adhered to its original objective of providing a ubiquitous platform and language for communication between consumer devices. Java supports APIs and products for electronic consumer device communications by providing:

- versions of its virtual machine that can run under strict memory constraints
- an operating system suited for small consumer devices
- APIs that support embedded devices and real-time operating systems (RTOS)
- tools that support software and hardware development in the consumer device arena

What makes Java the right choice for the consumer device communications market? Well, Java is inherently distributed, has rich support for networking and multithreading, promises the much desired "platform independence," and allows development in a common high-level object-oriented environment.

This month we'll peer into the Cosmic Cup's crystal ball to identify APIs and products defined by Sun that will further the cause of Java for consumer devices. Please note that while there's a wide range of embedded products and APIs available from several vendors, this month's column focuses on those provided by Sun because they're base products closely linked to the APIs defined under the scope of the Java Platform.

Consumer Devices APIs and Products

The APIs and products described below support the penetration of Java into the consumer device market. Figure 1 illustrates these products and APIs, and the consumer devices they support; Table 1 provides brief descriptions. The next section discusses the following consumer device products and APIs:

- JavaOS
- JavaCard
- EmbeddedJava
- PersonalJava
- Personal WebAccess

- Java Embedded Server

JavaOS

JavaOS is a distributed platform that's optimized to run on a variety of computing and consumer platforms without requiring the presence of a host operating system. JavaOS can directly execute the Java Virtual Machine (JVM) across a variety of hardware products, devices and CPUs.

Currently JavaOS is designed to support three industry segments:

- **JavaOS for Business:** An operating system software that leverages Java technology to define the next step in centrally managed thin-client computing. Its salient features are a functionally rich operating system for network computer manufacturers, solutions providers and hardware vendors; a new platform for Java technology-based applications written in the Java programming language; and a network computing solution for delivering business applications. It is a joint venture of Sun and IBM.
- **JavaOS for Consumers:** An optimized Java computing platform for consumer communication appliances such as Web phones, set-top boxes and handheld computing devices. It's a real-time operating system that provides a streamlined implementation of the PersonalJava environment. The product is offered by Sun.

API/Product	Description
JavaOS	A Java-based operating system that's optimized to run on a variety of desktop computing and consumer platforms without requiring a host OS
JavaCard	A Java Application Environment (JAE) that implements APIs that enable smart-card transactions via card-reader terminals
EmbeddedJava	A JAE for dedicated, embedded devices designed specifically for severely resource-constrained environments
PersonalJava	A JAE specifically designed for building network-connectable applications for consumer devices for home, office and mobile use
Personal WebAccess	Offers a compact Web browser for devices that run on the Personal Java platform; supports Internet standards including HTML 3.2, Frames, Tables, Cookies
Java Embedded Server	A small-footprint server product for use within remote embedded devices

Table 1: Consumer devices APIs and products

- **JavaOS for NCs:** A small, efficient stand-alone Java application platform designed specifically for network computers. It enables users to log in anywhere on the network and still use their familiar workspaces. This product is also offered by Sun.

The JAEs, namely JavaCard, EmbeddedJava and PersonalJava, provide different levels of Java operating system support and target a range of environments from smart cards to enterprise servers. These three, coupled with the full-featured Java Platform, provide the first upwardly binary-compatible platform that targets various segments of the enterprise today. Figure 2 illustrates these JAEs. Applications developed in the JAEs are upwardly compatible, that is, JavaCard applications will run in the EmbeddedJava JAE, which in turn is upwardly compatible with the PersonalJava JAE, which is upwardly compatible with the Java JAE.

JavaCard

JavaCard, the JAE with the smallest footprint offered by Sun, enables Java technology to run on smart cards and other devices with limited memory. The JavaCard API allows platform independence between different JavaCard-enabled platforms. The API defines the calling conventions by which an applet accesses the JavaCard runtime environment and native services.

JavaCard consists of a JVM, an API for smart cards and a framework that provides system services for smart-card applications. The minimum system requirement is 16 KB of ROM, 8 KB of EEPROM and 256 bytes of RAM.

The JVM layer hides the manufacturer's proprietary technology with a common language and system interface. JavaCard applications are basically Java applets. Multiple applets may reside on a single smart card.

The JavaCard Application Environment (JCAE) is licensed on an OEM basis to smart-card manufacturers. Some vendors have announced the availability of JCAE, and the 2.0 specification was recently released.

EmbeddedJava

EmbeddedJava is a JAE designed specifically for severely resource-constrained environments in the embedded device market. It may be used to develop a variety of embedded products.

EmbeddedJava applications run on top of an RTOS dedicated to the embedded device for which the product is being developed. Such applications may be ported to other RTOSs. These products can scale from dedicated applications limited to a

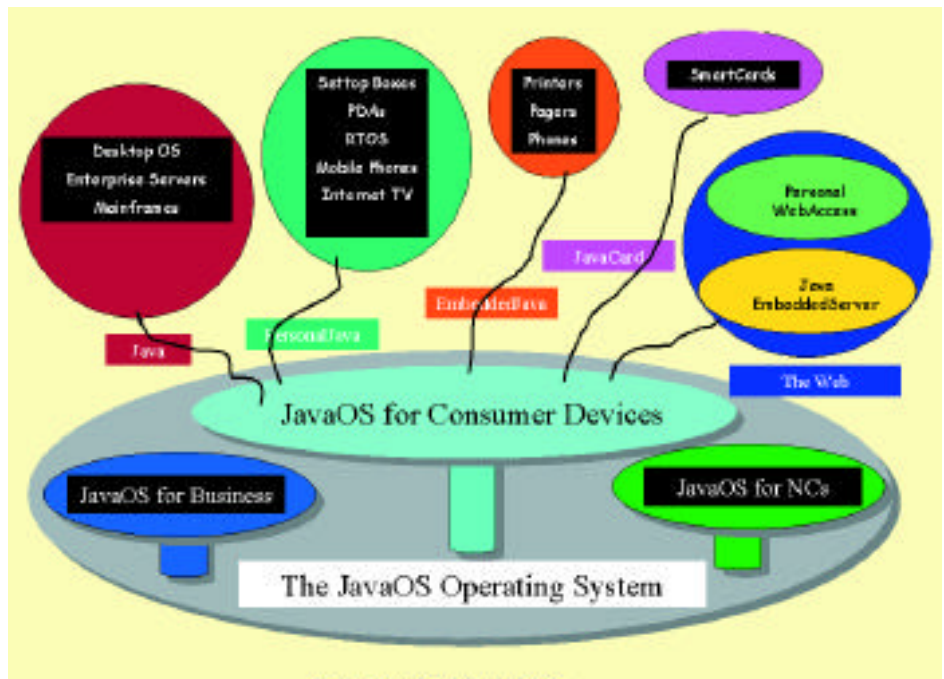


Figure 1: Java for Consumer Devices

device's ROM to complex, network-aware devices. Some products that can be developed using this JAE are mobile phones, pagers, process control, instrumentation, office peripherals, and networking routers and switches.

The API consists of a collection of configurable Java classes that have been reimplemented for memory-constrained dedicated embedded devices. The Java classes offered by EmbeddedJava may be categorized into two groups:

1. APIs derived from the core JDK classes, usually supplied directly by Sun, or via the device manufacturer or ISV. These classes include those dedicated to specific applications as well as JDK classes that are optimized to run in memory-constrained environments.
2. Hardware-specific classes, usually supplied by the hardware vendor

In addition to the API, the EmbeddedJava JAE consists of a JVM (supplied by Sun

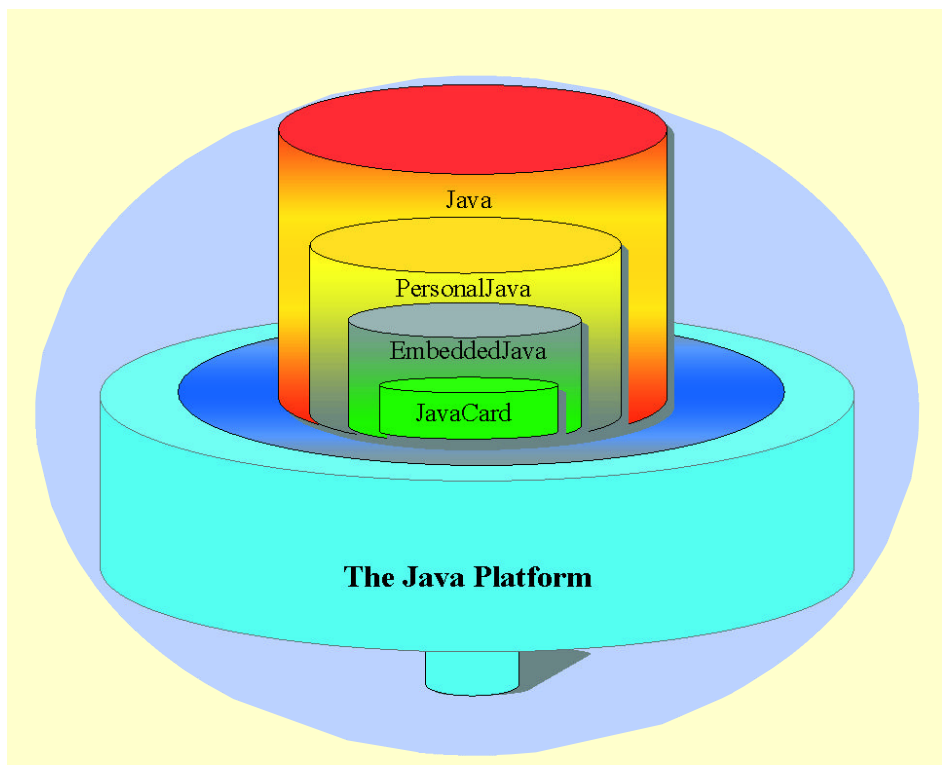


Figure 2

1/3 Object Matter

or a reseller), an RTOS (supplied by a third-party RTOS vendor) and device hardware (supplied by the device manufacturer).

The EmbeddedJava specification is open for public review.

PersonalJava

PersonalJava is a JAE specifically designed for building applications for network-aware consumer devices such as mobile handheld devices, set-top boxes, game consoles and smart phones. It consists of a JVM and a subset of the JDK APIs including core and optional APIs as well as the class libraries. These APIs are supplemented by a small number of new APIs designed to meet the needs of network-embedded applications. PersonalJava shares a common core set of APIs with the EmbeddedJava API. It also supports the environment to run Java applets.

The minimum system requirements for a device running PersonalJava are a 32-bit CPU with 50+ MHz of memory, 2 MB of ROM, 512 KB of RAM, keyboard or alternate input devices and a mechanism for downloading applets.

PersonalJava is commercially available as version 2.0.

Personal WebAccess

I discussed Personal WebAccess, a customizable, compact Web browser for devices running the PersonalJava platform, in last month's column and therefore won't repeat myself here.

Java Embedded Server

Java Embedded Server (also known as Project NanoServer) is a new type of Web server suited to the network-aware consumer device market. It runs on the device itself and manages its services.

Java Embedded Server targets embedded devices, which have a wide range of use and span several industry segments. Smaller devices like PDAs, cell and Web phones, routers, switches and set-top boxes can run the Java Embedded Server. At the same time, the server is at home on televisions, automobiles and manufacturing equipment as well as on office equipment like copiers and fax machines.

The server is dynamic in nature. It can be configured and customized at real time. Services can be added and removed, built new and even downloaded from a remote location on the network. It consists of two parts:

1. **The JavaServer engine:** Comprises APIs for life-cycle management of network-pluggable devices and applications. With its very small footprint (100 KB) it provides installation, versioning, content

management, recovery and service, for example.

2. **JavaServer services:** Invoked and managed by the Java Embedded Server, they include HTTP, SNMP, logging, threading, remote administration, RMI and servlet support. They also include services like Mail and Fax that may be built on top of the JavaServer services.

The Java Embedded Server is in its beta stage and its current working name is Project NanoServer.

For Details . . .

Links for detailed information on these products may be obtained from Sun's Java Web site (<http://java.sun.com/products>).

This article concludes the whirlwind tour of JavaSoft's APIs and products that are defined under the scope of the Java Platform. In the last five articles our Cosmic Cup has provided an eagle's-eye view of the Java world. This in no way completes the APIs and products defined for Java. Even as I write, new APIs and products are being defined under the Java Platform. And several other sectors in the industry are also involved in defining Java, both the language and the platform.

In future articles we'll examine the APIs and products developed by other industry players involved in the Java Platform's evolution. We'll also take a closer look at the individual pieces of the Platform.

Cosmic Reflections

The volume and maturity of the hardware environment far exceeds the progress made by software solutions. Giving something away for free doesn't earn the big bucks for any company. However, giving a language away free as a means to an end will eventually pay handsomely. The consumer devices market is massive in terms of sheer volume. In this new computing paradigm, combining the distributed networking facilities and platform independence offered by Java with the embedded chip market is sure to lead to a revolution in the enterprise. ●

About the Author

Ajit Sagar, a member of the technical staff at i2 Technologies in Dallas, Texas, holds a BS in electrical engineering from BITS Pilani, India, and an MS in computer science from Mississippi State University. Ajit focuses on networking, UI and middleware architecture development. He is a Java-certified programmer with eight years of programming experience, including two in Java. You can e-mail him at ajit_sagar@i2.com.



Ajit_Sagar@i2.com



IIOOP Explained

It may be the next universal Internet protocol

by Michael Barlotta

The Internet is reshaping both the business and computing worlds, defining new ways in which business is done and how applications are designed and developed. The Web allows businesses to build distributed applications that enable the sharing of information around the world and both customers and employees to interact directly with business operations. Some strengths of the Web and Internet are their ability to provide uniform access to information through a consistent user interface and the capacity to allow applications to work together across multiple platforms. The standard technologies that make heterogeneous computing across a global network possible, including TCP/IP, HTTP and CGI, have fueled the rapid growth of applications on the Internet and intranets. However, as Web application functionality expands beyond static Web sites and basic data querying systems to more complex, mission-critical, enterprise Web applications, the current Web technologies become limiting factors in what can be achieved. To take full advantage of the Internet and the Web, a more robust, distributed-object, connection-based protocol needs to emerge as the standard communication protocol.

The Internet Inter-ORB Protocol (IIOOP) promises to do just that – unite objects and applications on the Web. IIOOP is an inter-ORB protocol and part of the CORBA specifications put forth by the OMG (Object Management Group). IIOOP has the potential to become the next standard communication protocol on the Internet, replacing, or coexisting with, HTTP/CGI. It provides an object-based protocol that could greatly enhance the types of applications that are built and placed on the Web. Browsers and Java applications could work together to “surf” distributed objects instead of just content. IIOOP has widespread industry support and is becoming a common feature in application server software and middleware. Should

IIOOP continue to garner support, it will become as common a protocol on the Web as TCP/IP and HTTP are now. This article describes IIOOP and how it could turn the Internet into a network of distributed, interoperable objects.

What Is CORBA?

Before diving into IIOOP, you need a brief overview of CORBA. CORBA is a distributed technology that supports access to remote objects developed in multiple languages across a variety of platforms, operating systems and networks. The CORBA 1.1 specifications spelled out the core functionality of an ORB (Object Request Broker), and described an IDL (Interface Definition Language) to define objects it interfaces with. The core of the CORBA architecture is the ORB, which is the object bus. The ORB allows client applications to find objects and invoke methods on them locally or across a network. It handles passing requests, responses and exceptions between a client object and a server object. When the client application uses an object, it doesn't need to know the object's location, programming language or type of platform because the ORB masks these details. The ORB handles the location of server objects in a repository that keeps this level of detail from the client. Objects the ORB will manage need to be registered so they are available to client applications.

Objects are the software solution to a business problem. They can be used to better describe and implement business processes. They're pieces of reusable software that encapsulate data and methods. The object class gives the object a name and an interface, and defines what it can do. An object class is used to build object instances in memory at runtime, and is written in a language such as Java. IDL can be used to describe the object class and methods in a language-neutral way. IDL doesn't actually implement any of the func-

HTTP, SMTP, SNMP...all those great four-letter acronyms name the application-level protocols that make the Internet click. In this issue Michael Barlotta delves deeper into the details than we usually do to explore the fast-growing Internet protocol, IIOOP, that makes CORBA work on the Web.

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

tionality an object will provide, but it can be used to create proxies that make distributing an object possible. It's also used to mask the language used to build the object from the client.

The client knows about server objects and functions through the proxy or stub generated by the IDL. A similar proxy or skeleton is generated for the server. These proxies handle marshaling, masking the fact that the object or request is remote. The proxies also manage object references that are used to identify a particular object instance of an object. The stub is a stand-in for the remote object on the client while the skeleton is a stand-in for the remote request on the server. The proxies allow each application to act as if the object or request came from within its own process (see Figure 1).

Proxies enable static calling of object methods and require the stub to be compiled with the client. In addition to using proxies, CORBA also allows clients to call objects using the Dynamic Invocation Interface (DII), which allows client applications to call objects of which it has no knowledge at compile time.

CORBA 1.1 outlined the workings of this object bus, ensuring that objects written for one ORB could be moved to another. It didn't guarantee that the ORB from one vendor could communicate with one from another vendor. The ORB vendors were free to

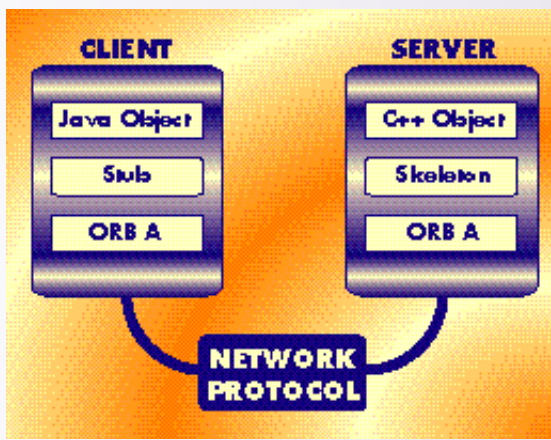


Figure 1

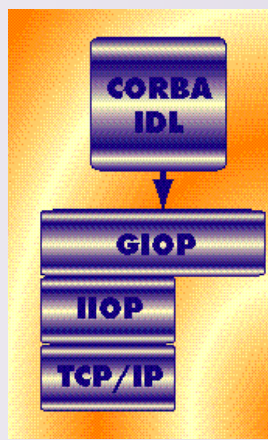


Figure 2

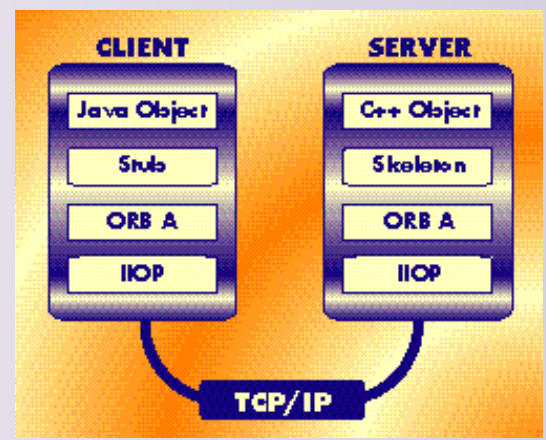


Figure 3

design their ORB products using any technologies and techniques they desired as long as they were compliant with the specifications. This resulted in several incompatible, proprietary ORB implementations. The CORBA 2.0 specifications addressed this concern, providing interoperability between ORB products through IIOB.

What Is IIOB?

CORBA 2.0 outlined a General Inter-ORB Protocol (GIOP) that specifies how ORB products from different vendors could communicate with each other. GIOP defines a standard data transfer syntax and messaging format over a transport protocol. It doesn't specify a particular communication protocol. It's a simple, protocol-neutral specification that can be mapped to several different network protocols. Since an ORB is concerned with maintaining object and data persistence between requests, GIOP assumes that the transport protocol is connection-oriented.

GIOP messages are made up of requests and replies in the traditional client/server model. To invoke a method, a client sends a request to a server, which waits for requests. On receiving one, the server processes it and sends a reply and any return values to the client.

The GIOP specification outlines the mapping of OMG IDL data types and object references to a common network representation known as CDR (Common Data Representation). CDR is an over-the-wire data format that handles marshaling of data types, byte ordering and other low-level operations to ensure that data is transported in a common standard format so that ORB invocations and information sent between machines on a network mean the same thing to both applications.

IIOB, a specific implementation of GIOP, defines how GIOP messages are mapped to the TCP/IP protocol (see Figure 2). To ensure interoperability between ORBs, CORBA 2.0

specifications dictate that all ORB products must support IIOB, either natively or through a bridge. Figure 3 illustrates how a client application can access an object on a different ORB through IIOB. Without IIOB to bridge the gap between incompatible ORB products, the client application using ORB A wouldn't be able to access the server object hosted by ORB B. IIOB provides a common interface that masks the ORB implementations behind it, making each ORB appear the same to each other.

How IIOB and TCP/IP Work

Most network stacks and communication protocols are compared to the OSI reference model when attempting to describe the role it plays in allowing communication between machines and applications. The OSI reference model, shown in Figure 4, has seven layers, each with a different set of responsibilities and its own set of APIs used to communicate with the layers above and below.

As a network message is received, it moves up the stack, allowing each layer to perform its tasks, such as routing, error checking and data marshaling. Each layer removes any message header information pertinent to its layer and passes the message along to the next layer. When a message is sent out on the network, the opposite occurs. The higher up the stack a protocol rests, the more abstract it is and the more low-level communication details it takes care of. The APIs used to interact with a protocol on higher levels mask the intricate details of network messaging and marshaling handled by lower layers, while the level of abstraction between layers allows protocols in the higher levels to support several different lower-level protocols. Using higher-level protocols makes application development easier and more portable. Leaving communication management to a protocol frees developers to develop application objects.

IIOB

IIOB is a high-level protocol that takes care of many of the services associated with the levels above the transport layer, including data translation, memory buffer management and communication management. It's also responsible for directing requests to the correct object instance within an ORB. An object instance is identified by an Interoperable Object Reference (IOR), which is specified by the GIOP and generated by the ORB. Because actual object references are handled differently by each ORB, an IOR is used to pass object references between different ORB products. The client application can access the object using the IOR, which masks the client application's ORB implementation from the ORB implementation used to host the CORBA object.

TCP

TCP (Transmission Control Protocol) makes up the transport layer of the stack and is responsible for ensuring that the right application on a machine receives the message. It's also charged with dividing messages into smaller packets and reassembling incoming packets. TCP uses a port number to identify a particular server application. Most "well-known" server applications and services have a default port number on which they listen for requests. For example, 80 is the default port number for an HTTP Web server. Using a well-known port number makes it easier for client applications to find servers they are interested in connecting to. Unfortunately, IIOB doesn't have a default port.

IP

IP (Internet Protocol) is a network layer protocol that provides inter-network communications and an addressing scheme: the familiar IP addressing that's used on the Internet and Web. An IP address is a 32-bit number, usually represented in dotted decimal notation, for example, 161.125.23.116.

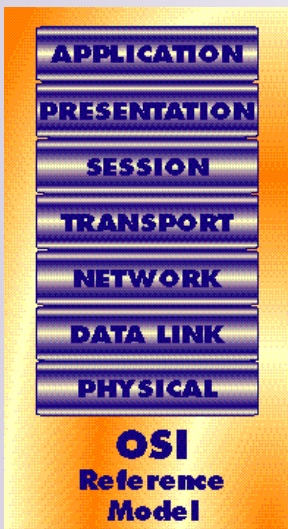


Figure 4

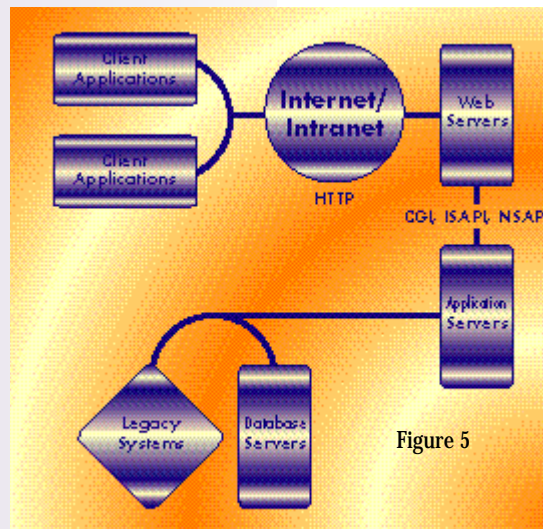


Figure 5

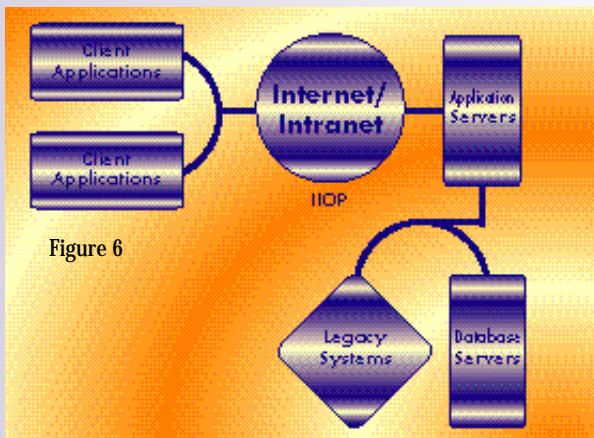


Figure 6

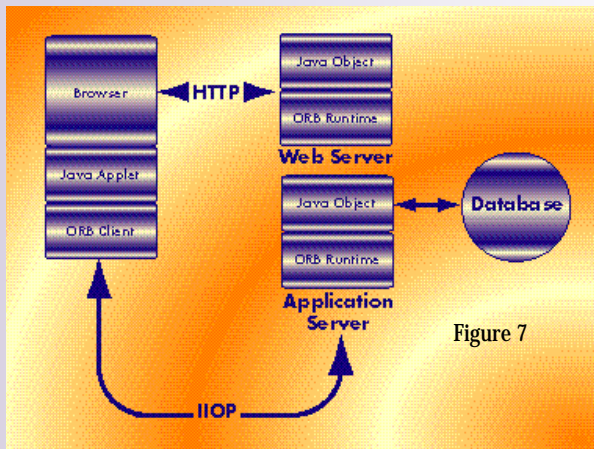


Figure 7

Each part of the address represents a byte of information ranging in value from 0 to 255. Depending on the class and subnet mask used by the IP address, a portion of the number represents the network ID and the remainder identifies the host or node on the network.

IIOOP and HTTP

The Internet is now dominated by HTTP/CGI applications. The Hypertext

TCP/IP, helping it fit right into the Internet and intranet environments. IIOOP provides a standard, robust protocol and, when coupled with the portability of the Java language, will eventually replace HTTP/CGI as the model on which Web applications are built.

IIOOP has the following advantages over HTTP/CGI:

- It doesn't require spawning a new instance of a program for each request.

- It supports more robust argument data types than strings.
- It provides the ability to obtain references to any objects on the Internet defined by OMG IDL.
- It provides persistence of state between calls.

In addition, CORBA ORB and application server products provide dynamic function invocation, load balancing, transaction services and other robust features not found in most CGI applications. Figure 6 provides an overview of an IIOOP-based Internet application.

IIOOP, Java and the Internet

The portability of the Java language and the openness of IIOOP can be combined to create new and powerful Web applications. Java applets can communicate with objects via IIOOP, allowing clients to access functionality without going through a Web server or using CGI. Java's portability rests on its ability to be compiled as bytecodes, which can be downloaded and run on a variety of platforms and operating systems using a Java Virtual Machine (JVM). JVM technology, available with most browsers, makes running Java applets and applications accessible to every client on the Internet. Java applet technology allows code to be deployed and managed from a central server, downloaded over the Internet and run on a client workstation. This enables changes to be made to Java programs and redeployed easily because each client downloads the code whenever it's needed.

Using a Java-enabled browser, a user can enter a URL to access a company's Web server. The Web server can return an HTML page and a Java applet over HTTP to the browser. The Java applet will contain an interface that allows the user to interact with the application. Using the client-side ORB, the Java applet can send requests to objects on another server using IIOOP. These objects will be hosted by an application server or ORB, and can provide access to various data sources and processing. Accessing objects over IIOOP instead of HTTP allows the Java applet to take advantage of the distributed object protocol, passing objects, various data types and maintaining state.

Figure 7 illustrates how the browser, Java, HTTP and IIOOP can work together to provide a solution to building Web applications, a solution made even easier because of the support and widespread availability of IIOOP by several leading software vendors. Netscape, Inprise/Visigenics and other application server vendors want to make CORBA and IIOOP as ubiquitous as HTML and HTTP are today. To make this a reality,

Netscape and Inprise/Visigenics are making Java and IOP available on the Web by bundling their technologies. Netscape Navigator 4 ships with a Java Virtual Machine and Visigenic's Visibroker for Java ORB, making Java and IOP available to every client that uses a Netscape browser.

In addition to being widely available on the client, application servers and ORBs are also providing support for IOP. The application server is a new breed of software that combines ORB functionality with that of a TP Monitor and Web server. Application servers host objects while providing scalability, reliability and access to various data sources in the middle tier.

Table 1 lists some of the major companies that support or plan to support IOP in their Web/Application server products.

It's interesting to note that the Visibroker ORB is licensed to SilverStream, Sybase, Oracle and Netscape, and is the basis for the CORBA/IOP support in their products. Sun has stopped supporting their ORB products, NEO and Joe, and is encouraging customers to migrate to Visibroker. In addition, Sun has announced that Java RMI will support IOP as well as JRMP as the underlying protocols. This makes Visibroker the leading ORB solution on the Internet.

The one notable company not listed as supporting IOP is Microsoft, which is pro-

Company	Products
Inprise/Visigenics	Visibroker
Netscape	ONE, Netscape Enterprise Server, Netscape Communicator
Sun/NetDynamics	NetDynamics
IBM	SOM, Component Broker
Iona	Orbix
Oracle	NCA, Oracle Web Application Server
SilverStream	SilverStream 2.0
Sybase	Jaguar CTS 2.0

Table 1

moting DCOM as an alternative protocol. DCOM is not widely supported by non-Windows platforms, however. Microsoft tools and servers, including Internet Explorer, IIS and MTS, aren't likely to support IOP in the near future. ORB client software, however, and products such as IOP Engine by Iona, will help client applications without Netscape's browsers interact with objects on the Web. Bridges between DCOM and

CORBA/IOP will also enable MTS and IIS to interoperate with objects over IOP on the server side.

Conclusion

IOP provides a solid basis for the development and deployment of enterprise Web applications. The distributed object support and functionality it provides go a long way in overcoming HTTP/CGI limitations. Used in conjunction with the Java programming language, portable distributed applications on the Web are becoming viable solutions to business problems. Despite competition from DCOM, IOP has both industry support and strong technology, making it the best candidate for becoming the next universal Internet protocol. ●

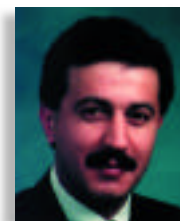
About the Author

Michael Barlotta is vice president of business development at MO Systems Solutions, Inc., and a CPD associate. He has a BA in computer science from The University of Albany (SUNY), New York. Mike is a contributing author to PowerBuilder 6: Secrets of the PowerBuilder Masters (**SYS-CON Publications, Inc.**) and the author of Distributed Application Development with PowerBuilder 6 (Manning Publications). You can contact him at mbarlotta@mosystems.com.



mbarlotta@mosystems.com

1/2 Ad



ROAD:BeanBox

by Specialized Software

BeanBox's capabilities are well worth it, especially for beginning programmers

by Ed Zebrowski



So there it is – the opportunity of a lifetime. It seems that the local public utilities commission has allocated a hefty budget to a new project. They want to be able to “link” all their users in the county utilities building to the same database. All they need is a good database administrator to come in and make it happen.

This appears to be a pretty cut-and-dried job on the surface, but when you arrive you are horrified to learn that all is not as it seems. Most of the office employees are on a Windows NT network, the engineers upstairs need their UNIX system and, to make matters worse, Bessie down in records in the basement won't part with her Mac. (This truly is a government operation!)

The development of database applications that must run on many different platforms points to one solution: Java! This is okay if you're a proficient Java programmer, but what if you're not? Fear not! Your problem is solved with ROAD:BeanBox from Specialized Software International.

ROAD:BeanBox creates Java applications on the fly that will access data from JDBC-enabled database servers. It contains a rich set of data-access components and data-aware user interface (UI) components implemented in Java, using JavaBeans 1.0 technology. Best of all, advanced programming knowledge isn't required to use it.

Installation of ROAD:BeanBox

ROAD:BeanBox runs on any JDK1.1 Virtual Machine or browser. The actual installation procedure varies depending on the software that's to be used with it. I experimented with ROAD:BeanBox in two ways. I tried using it with the JDK1.1.5 and then with Visual Café.

The version of ROAD:BeanBox I used was packaged on CD-ROM as a JAR file. When using the product with JDK 1.1.5, installation consisted of the following steps:

- Make a directory on the hard drive with an appropriate name. (I called mine BeanBox, of all things!)
- Copy the appropriate JAR file from the CD-ROM to that directory. In this case the file copied is called “bbox.jar”.
- Ensure that the CLASSPATH variable has been modified to point to this JAR file (for example: CLASSPATH=%CLASSPATH%;C:\BeanBox\bbox.jar).

A great feature of ROAD:BeanBox is that, in addition to the JDK, it works with some GUI-based IDEs as well. I had pretty good results using it with Visual Café. To install ROAD:BeanBox through Visual Café, follow these steps:

- Make sure Visual Café is installed and operating properly.

ROAD:BeanBox 1.1

Specialized Software International, Inc.

120 Stafford Street

Worcester, MA 01603

Phone: 800 328-2825

Fax: 508 754-8973

E-mail: info@specializedsoftware.com

Web: www.specializedsoftware.com

Price: \$295 for a single-user developer license (\$395 for a single-user developer license with one-year subscription)

- Launch Visual Café.
- Create a new project by selecting the “Empty Project” template from the “New Project” dialog box.
- From the “Insert” menu select the option “Component into Library.”
- Select the bbox.jar file. Click “OPEN” to begin importing.

What's New in Version 2.0 of ROAD:BeanBox

For Enterprise and advanced programmers, especially JDO objects

Since the review of version 1.1 of ROAD:BeanBox, Specialized Software has released version 2.0, improving the features and adding many more beans for the same low price. Some of the important additions and improvements include:

Java Data Objects (JDO)

A complete set of data objects called Java Data Objects. JDO objects are JavaBeans that can be independently created, allowing users the ability to track only those objects they need. The improved client-batch cursor library and full connectionless result set allow easy migration of data from one database to another. Direct access to stored procedures, connection pools, improved error handling, cursor-enabled meta objects, database events and thread safety are just some of the main features of JDO objects. JDO objects are best used in building server-side Java applications.

Data Accessor Object (DAO)

Another key component in ROAD:BeanBox v2.0 is the Data Accessor Object bean. This GUI bean provides easy access to data stored in databases using JDO objects. DAO features an easy-to-use customizer to set properties and includes automatic data binding support for AWT and JFC components, including the JTable. DAO supports bidirectional data navigation and includes a database message-aware StatusBar, plus data-aware AWT components such as MaskText, Label, etc.

JGrid – Sophisticated Grid Bean

A new user interface bean called JGrid presents data in a two-dimensional table format. JGrid provides full printing support, smooth scrolling and support for multiple headers. The JGrid design is based on the model view controller architecture. JGrid is bundled with Cell Renderers (Text, Image, CheckBox, RadioBox, etc.) and CellEditors (Text, MaskText, DropDown List, Calendar, etc.).

- From the "Tools" menu select "Environment Options."
- Switch to the "Component Palette" tab.
- Select ROAD:BeanBox from the "Available components" list and click on the "Add" button.
- Click the "OK" button to save the changes.

The Application Wizard: Your Key to Quick Results

At this point I'd like to strongly recommend the installation and use of the Application Wizard. ROAD:BeanBox seems to work best through the wizard. To install this Wizard when working with JDK1.1.x, take the following steps:

- Be sure you have properly installed Road:BeanBox.
- Copy the file "appwiz.jar" to the directory you created when originally installing ROAD:BeanBox.
- Change your CLASSPATH variable to point to this file. Mine looks like this: CLASSPATH=%CLASSPATH%;C:\BeanBox\bbox.jar;C:\BeanBox\appwiz.jar.

Installation of the Application Wizard through Visual Café is much the same as it is with the JDK:

- Create a "BeanBox" directory on your hard drive.
- Copy both the bbox.jar and the appwiz.jar files to this directory.
- Modify your CLASSPATH environment to point to these files: CLASSPATH=%CLASSPATH%;C:\BeanBox\bbox.jar; CLASSPATH=%CLASSPATH%;C:\BeanBox\appwiz.jar.

At this point it's necessary to create a link to an ODBC data source with a DSN name pointing to that source. The method used here will vary depending on the database you're working with. To keep things simple, I'll run through the sample supplied with my copy of ROAD:BeanBox. It's a Microsoft Access file named "admin.mdb." The sample is a good way to get a feel for this product. To run the sample take the following steps:

- Be sure that the "Microsoft Access ODBC driver" has been properly installed and configured.
- Copy the admin.mdb file to the BeanBox directory created during installation.
- Run "ODBC" from the control panel, adding a new Data Source sample. When prompted, specify the following attributes:

ODBC driver = Microsoft Access Driver
Data Source = sample
Login name = prasad
Password = doll1102
Database=C:\BeanBox\admin.mdb
(Or whatever the absolute path to the file is on your system)

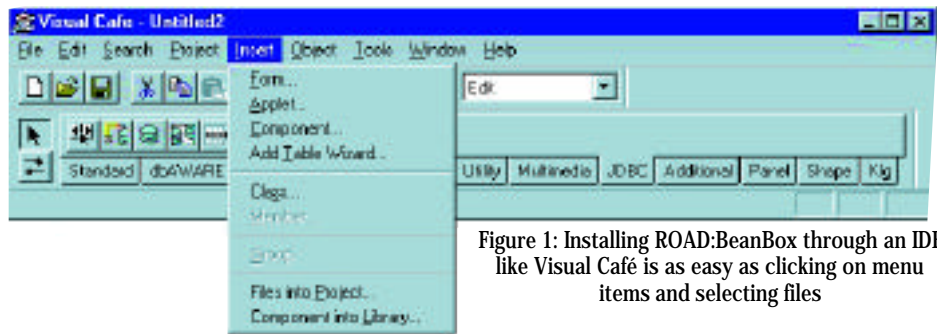


Figure 1: Installing ROAD:BeanBox through an IDE like Visual Café is as easy as clicking on menu items and selecting files

The Application Wizard is initiated with JDK or Visual Café with the same command: java TableAppWizard.TableAppWizard.

If all goes well, this will bring up the welcome screen to the Application Wizard. Clicking "Next," you will then come to the login screen. Here you must specify the data source name you created during ODBC setup. In our example, the information entered is:

DataSource= Sample
SubProtocol=odbc
UserName=prasad
Password=doll1102

Clicking the "Next" button establishes the connection to the database. The "Customized Info" screen will now display. Entering "Employee" as Application Name will change the directory path to c:\employee.java. For Application Format select the radio button named "Single Row - Vertical." Clicking the "Next" button takes you to the "Table-Column Information" screen. Select the table "PRASAD.DEMO_EMPLOYEE." This will automatically change the index name to the appropriate SQL number. The Columns List box will be filled with the columns in the PRASAD.DEMO_EMPLOYEE table. Click the ">>" button to select all the columns for display in the generated application. When you press the "Next" button you should get a pop-up box confirming the launch of the Visual Designer. The Visual Designer is used to preview and/or modify the layout of the screen before the code is generated. The fields can be resized by clicking and dragging to suit your needs. Pressing "OK" will generate the code generation confirmation box. Then press "OK."

Now it's time to compile the code you've generated. In JDK this is done with the command javac c:\Employee.java. This class file runs with java c:\Employee.

If you're using Visual Café, import the generated code into the project. Set the main class parameter of the "Project" tab to "Employee." You will now find the "Project" tab in the "Project Options" box. Build and execute the application according to the documentation supplied with Visual Café.

As you can see, it's possible to create a

single table maintenance screen in a matter of minutes. It's also possible to create "multiple row" single-table applications as well as "master detail relationship" applications using ROAD:BeanBox. Although these are a little more involved, they come much easier and quicker than with most other methods. This benefits the user because it's not necessary to take the time to learn how to write complicated programs to access and manipulate data from a server. It also allows a significant reduction in testing and configuration time. Since the amount of code is less, the amount of time testing and adjusting the code is significantly reduced. The ability to perform database tasks with less code provides other benefits as well:

- Reduction in maintenance overhead. It's logical - less code equals less maintenance.
- No more duplicate database connections. ROAD:BeanBox's Connection Sharing capability centrally manages database connections to minimize the use of costly database resources.
- Improved performance. Road:BeanBox is equipped with a backward scrolling feature for static cursors. Coupled with the batch update feature for batch cursors, this will improve response time and reduce network traffic.
- Scalable and extensible platform. The user is provided with reusable classes, all of which have specific functions. These classes can easily be extended for the use of many business requirements.

ROAD:BeanBox is one of the neatest development tools I've seen. Although it has a full arsenal of powerful database access tools, you need only a very basic knowledge of the Java programming language to use it effectively. That alone makes it worth its weight in gold. ☘

About the Author

Edward Zebrowski is a technical writer based in the Orlando, FL area. Ed runs his own Web development company, ZebraWeb.



zebra@rock-n-roll.com



Applet and Servlet Communication

Sending a serialized object

by Chad Darby

Java servlets provide a new way to develop server-side solutions. They provide the features of traditional CGI scripts with the additional benefits of efficiency and portability. Currently, major corporations are making the migration from CGI scripts to Java servlets. As a result, the demand for applet and servlet communication is on the rise.

In the February 1998 issue of *JDJ* (Vol. 3, Issue 2), I presented a three-tier database application that used Java servlets. In this article you will learn how to build a three-tier database application that allows a Java applet to perform two-way communication with a Java servlet. I'll focus on the concepts and techniques of applets communicating with servlets, building on the application presented in the previous article. Don't worry if you missed that article; a review follows.

Reviewing Our Student Tracker Application

The previous article presented a three-tier database application that used Java servlets and the Java Database Connection (JDBC). That application allowed a public speaker to keep track of students who attended the seminars. Students interacted with the application by entering their contact information into an HTML form. Once the form was submitted, the Java servlet used JDBC to store the student information in a database. Afterwards, an updated student list was generated by the servlet and returned to the user as an HTML page.

The application was partitioned into three tiers: user interface layer, the business rules layer and the DataStore layer. Figure 1 illustrates the design.

The first tier is a Web browser, which serves as our universal client. In the first phase of the application, an HTML front end was used for user input and to display the database query results. The HTML approach was taken because it lowered the

requirements of the client's Web browser version. This low-tech approach made the application accessible to users whose browsers were not Java 1.1 enabled.

The second tier of the application was implemented with a Web server capable of executing Java servlets. The Java servlet harnessed the power of JDBC to access the database to store/retrieve information as needed. A dynamic HTML page was generated by the servlet based on the database results.

The third tier was composed of our back-end database server, which stores the information used by the application. Thanks to the JDBC API, however, the servlet can access the database in a portable fashion by using the SQL call-level interface.

Developing an Applet Front End

To enhance the student tracking system, we will develop an applet front end. Students can now enter their contact information into a Java dialog box. Also, an updated student list is displayed in a Java list component. Figure 2 shows the new applet front end.

Applet-Servlet Communication with HTTP GET and POST

In the previous version the HTML form was used to submit the student's data to the servlet. Accessing the form data on the server side was simple and straightforward. This was accomplished by calling the method `HttpRequest.getParameter(<form field name>)`, which is available in the Java servlet API.

We are now using an applet front end and we need a mechanism for the applet to communicate with the servlet. We need to capture the information a student enters and somehow pass it to the servlet. Since servlets support the HTTP/CGI interface, we can communicate with the servlet over HTTP socket connections. The applet sim-

ply opens a connection to the specified servlet URL. Once this connection is made, the applet can get an output or input stream on the servlet.

The applet can send data to the applet by using a GET or a POST method. If a GET method is used, the applet must URL-encode the name/value pair parameters into the actual URL string. For example, if we wanted to send the name/value pair of `LastName=Jones`, our servlet URL would resemble:

```
http://www.foo.com/servlet/TestServlet?LastName=Jones
```

If you have additional name/value pairs, they are separated by an ampersand (&). If we add a name/value pair of `FirstName=Joe`, our revised servlet URL would resemble:

```
http://www.foo.com/servlet/TestServlet?LastName=Jones&FirstName=Joe
```

We would have to URL-encode each name/value pair for the student's contact information. To send a GET method to a servlet, the applet can use the `java.net.URLConnection` class. The code fragment below shows you how.

```
String location =
"http://www.foo.com/servlet/TestServlet?LastName=Jones";
```

```
URL testServlet = new URL( location );
URLConnection servletConnection = testServlet.openConnection();
```

```
InputStreamFromServlet =
servletConnection.getInputStream();
```

```
// Read the input from the servlet.
...
```

Once the applet has opened a connection to the URL, the input stream from the servlet is accessed. The applet can read this input stream and process the data accordingly. The type and format of the data returned depend on the servlet. If the servlet is returning custom information, a custom messaging protocol has to be creat-

ed for the applet and servlet to communicate. I won't go into the details of a custom protocol, however, as I'll present an elegant solution later in the article.

To POST data to a servlet, the `java.net.URLConnection` class is used again. This time, however, we must inform the URL connection that we will send data over the output stream. The POST method is powerful because you can send any form of data (plain text, binary, etc.). All you do is set the content type in the HTTP request header. The servlet must be able to handle the type of data that the applet sends, however.

The code in Listing 1 shows how to send a POST method to a servlet URL. The details of transmitting the data are discussed later in the article.

As you can see, applets can communicate with servlets using the GET and POST method.

Communicating with Object Serialization

In our application we want to provide a higher level of abstraction. Instead of passing each parameter of student information (e.g., last name, first name) as name/value pairs, we'd like to send it as a true Java object. Our Java application already has a `Student` class that encapsulates all of the information about a student (see Listing 2). This information is gathered from the New Student dialog box and a `Student` object is created. When we register a new student, we'd simply like to send the `Student` object to the servlet. Upon receipt of the `Student` object, the servlet would add the new student to the database. We also want the servlet to send the applet an updated student list as a vector of student objects. This will allow the applet to display the student list quickly and easily.

You may ask how we can accomplish this. It's easy, thanks to Java 1.1's object serialization, which allows an object to be flattened and saved as a binary file. The values of the data members are saved so the state of the object is in fact persistent or serialized. The object can be loaded or deserialized later from the binary file with the values of its data members intact. Object serialization is fascinating because it frees the developer from the low-level details of saving and restoring the object.

How does this relate to applet-servlet communication? Well, object serialization is not limited to binary disk files. Objects can also be serialized to any output stream. This even includes an output stream based on a socket connection. So you can serialize an object over a socket output stream! As you've probably guessed by now, a Java

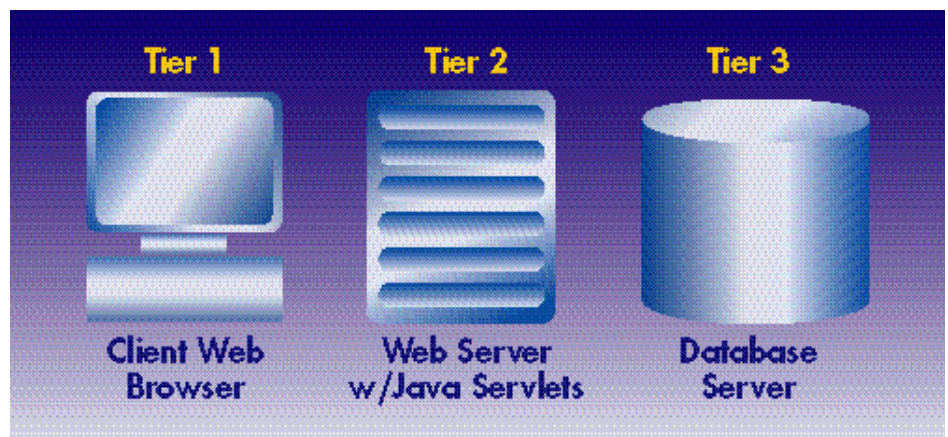


Figure 1: Three-tier design

object can also be deserialized or loaded from a socket input stream.

For a Java object to be serializable, its class must implement the `java.io.Serializable` interface. You won't have to actually implement any methods for this interface, however, because the interface is empty. The `java.io.Serializable` interface is simply a tag for the Java Virtual Machine. We can create a custom class as follows:

```
class Foo implements java.io.Serializable
{
    // normal declaration of data members,
    // constructors and methods
}
```

The following code fragment shows you how to serialize an object to an output stream. In this example we already have a socket connection to a host machine and are simply serializing the object, `myFoo`.

```
outputToHost = new ObjectOutputStream(host-
Connection.getOutputStream());

// serialize the object
Foo myFoo = new Foo();
outputToHost.writeObject(myFoo);

outputToHost.flush();
outputToHost.close();
```

Notice in this example that an `ObjectOutputStream` is created. This class is responsible for serializing an object. The object is actually serialized when the `writeObject()` method is called with the target object as its parameter. At this time a binary image of the object is written to the output stream. In this case the output stream is based on a socket connection.

This example wouldn't be complete, however, without code on the host machine to read the serialized object. The next code fragment shows you how to deserialize an object from an input stream.

```
inputFromClient = new
ObjectInputStream(clientConnection.getInput
Stream());

// deserialize the object, note the cast
Foo theData = (Foo) inputFromClient.readOb-
ject();

inputFromClient.close();
```

An `ObjectInputStream` is created based on the client's socket connection. The object is deserialized by simply calling the `readObject()` method. But we must cast the object to its appropriate class, in this case the class `Foo`. At this point the object is available for normal use.

As you can see, object serialization is very straightforward and easy. Now we'll use the technology to pass objects back and forth between our applet and servlet.

Sending Objects from an Applet to a Servlet

With the information presented so far, we can send a Java object to a servlet. In our Student Tracking application the applet sends a `Student` object to the servlet when a new student is registered. Figure 3 displays the object interaction between the servlet and the applet.

The code fragment shown in Listing 3 is used by the applet to send the `Student` object to the servlet. The applet is actually sending a POST method to the servlet. This client-side code fragment opens a URL connection to the servlet URL. We inform the servlet connection that we're sending output data over the connection and receiving input. Other methods are called so the connection will not use cached versions of the URL. An important call in this code fragment is `setRequestProperty()`. This method sets the content type in the HTTP request header to the MIME-type `application/octet-stream`, which allows us to send binary data, in this case our serialized `Student` object. The next couple of statements cre-

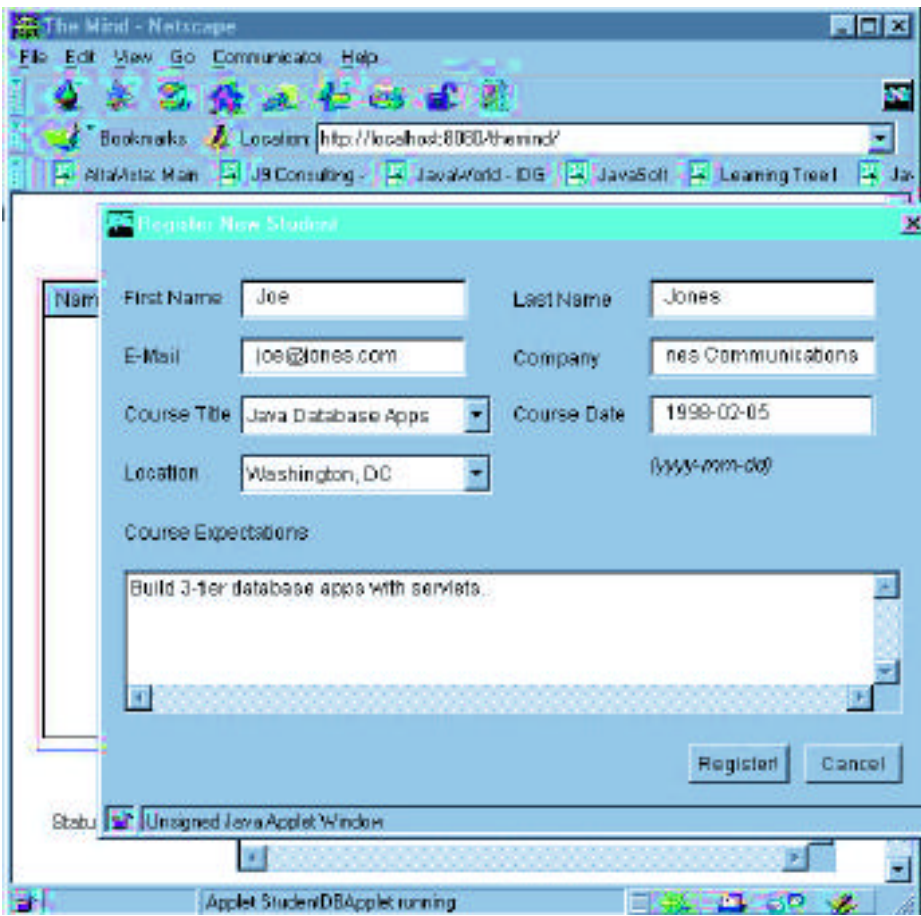


Figure 2: Student tracker applet

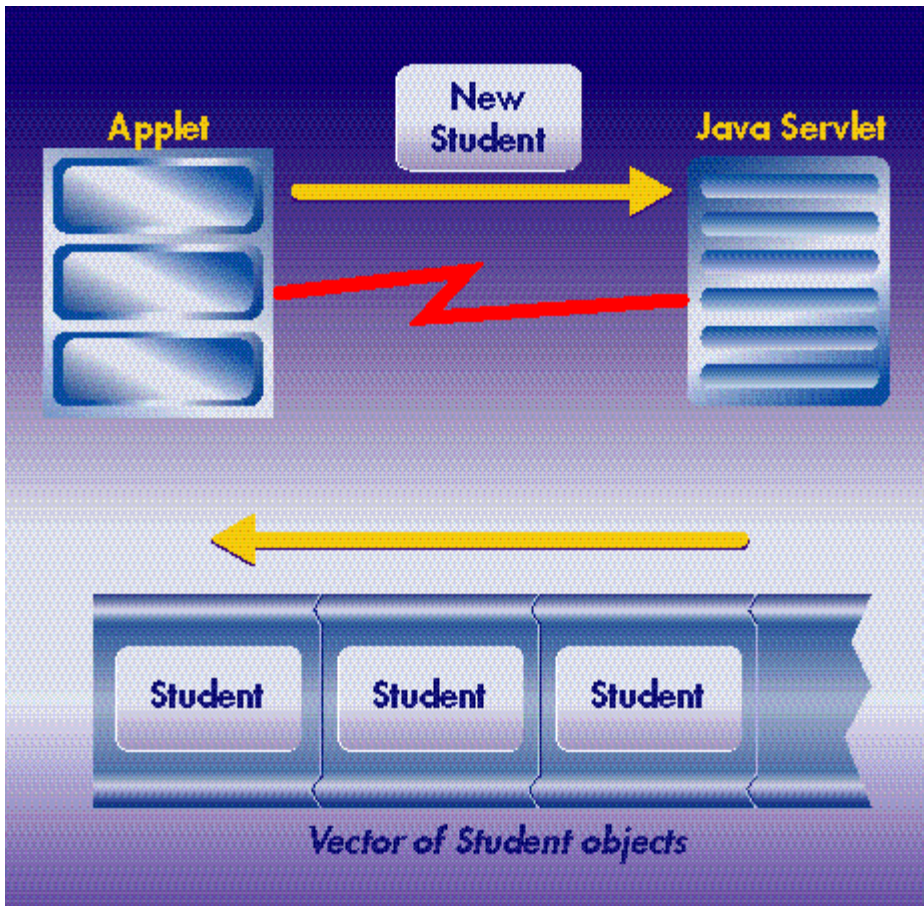


Figure 3: Applet-servlet object transaction

ate an `ObjectOutputStream` that actually writes the object to the connection stream.

We're not finished yet. Recall that our application is in the process of registering a new student. The servlet has to read this student object and update the database accordingly. Thus we need code on the server side to receive a serialized `Student` object.

The code fragment in Listing 4 displays the servlet code for reading a `Student` object from an applet. The servlet handles `POST` methods by implementing the `doPost()` method and acquires an `ObjectInputStream` from the requesting applet. From there it is simply a matter of reading the `Student` object from the stream. At this point the `Student` object is loaded and available for registration in the database. Please note the small number of statements on the server side for reading in a serialized object. You have to agree it's quite simple and straightforward.

Sending Objects from a Servlet to an Applet

The servlet in our Student Tracking application is now capable of receiving a student object and registering it in the database. Now the servlet has to return an updated list of registered students that is then returned as a vector of student objects. This interaction is also illustrated in Figure 3.

When the servlet returns the vector, there's no need to iterate through the vector and serialize each `Student` object individually. The servlet can serialize the entire vector in one step since the class `java.util.Vector` also implements the `java.io.Serializable` interface.

The code fragment shown in Listing 5 is used by the servlet to send a vector of `Student` objects to the applet. The `sendStudentList()` method is passed to an `HttpResponse` parameter and a vector of `Student` objects. Since the applet initiated the `HttpRequest`, the servlet can respond to the applet by using the `HttpResponse` parameter. Thus an `ObjectOutputStream` to the applet is created based on the `HttpResponse` object. The student vector is actually serialized and sent to the vector with a call to `outputToApplet.writeObject(studentVector)`.

As we've seen before, code is needed by the applet to handle the data sent from the servlet. The applet uses the code fragment shown in Listing 6 to read in a vector the `Student` objects from the servlet. The applet opens a URL connection to the servlet's location. Appropriate methods are called to ensure that the applet doesn't use cached versions of the URL connection. Next, an `ObjectInputStream` is created based on the servlet's input stream socket connection. All the switches have been flipped now and we can easily read in our

vector of Student objects. Again, remember that we have to cast the object to the appropriate type.

Congratulations! You've successfully read in a vector of student objects. This vector is now available for refreshing the AWT List component.

Conclusion

This article goes beyond the normal method of sending name/value pairs over the HTTP/CGI protocol. The techniques presented leveraged the features of Java object serialization, providing an elegant way to transmit serialized Java objects over network connections.

I must inform you, however, that this article discussed communication using the HTTP/CGI protocol only. There are a number of other mechanisms for applets to communicate with server-side processes – Java's Remote Method Invocation (RMI), for example.

RMI allows a client application to call methods on a remote object as if the object was local. In fact, RMI uses object serialization to pass objects back and forth between the client application and the remote object. All the low-level details of network connections and serialization are hidden from the developer using RMI. If your project requires a large amount of applet-servlet communication, I'd recommend

“Java's object serialization allows an object to be flattened and saved

that you take a close look at RMI and the features it has to offer.

The second mechanism of communicating with server-side process is CORBA (Common Object Request Broker Architecture). Like RMI, CORBA allows you to make method calls on remote objects. If you have legacy server-side code written in a different language, you can wrap it as a CORBA object and expose its functionality. CORBA provides a rich framework of services and facilities for distributing objects on the network.

By now, you should understand the concepts and techniques for communication between applets and servlets. If you'd like to get more information on distributed computing with RMI and CORBA, visit the Web sites listed below. In this article I demonstrated how an applet uses a POST method to send a serialized object to a servlet. The appropriate server-side code for the servlet was provided for reading in a serialized

object. Our Student Tracking applet used this communication method to send a true Java object to a servlet. The servlet was also enhanced to return a vector of student objects to the applet. Likewise, the appropriate applet code was provided for reading in a vector of student objects.

As you can see, applet and servlet communication is straightforward with the techniques presented in this article. You can now add an applet front end to your servlet-based application. ●

URL References:

Remote Method Invocation (RMI)

www.javasoft.com/products/rmi

CORBA: www.omg.org

Student Tracker Source Codewww.j-nine.com/pubs/applet2servlet

www.j-nine.com/pubs/applet2servlet

▼▼▼▼ CODE LISTING ▼▼▼▼

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

About the Author

Chád (shod) Darby, a Java consultant for J9 Consulting, specializes in developing server-side Java and database applications. You can e-mail him at darby@j-nine.com. Carnegie Mellon University.



darby@j-nine.com

1/4 Ad



SYS-CON Radio Host Robert Diamond with Bill Carson of ServerLogic

TUNE IN TODAY!



Hear Live Interviews with the Major Technology Vendors from the Java Business Expo at www.SYS-CON.com

JDJ Online

ne Spread

KL Group Launches New JavaBeans for the Enterprise

(Washington, DC) – KL Group, Inc., has announced the release of JClass 3.5, a new version of its popular family of JClass JavaBeans providing powerful new databound components.

Automatic databinding is now built into many of the JClass components, and this release introduces an Enterprise Suite and two new JavaBeans, JClass HiGrid and JClass DataSource, that let

Java developers build complete database applications without writing a single line of code.

For more information, visit KL Group's Web site at www.klg.com, or call Lee Garrison at 416 594-1026, ext. 545, or e-mail him at lee@klg.com.

Sun and IBM Introduce JavaOS

(Palo Alto, CA) – JavaOS for Business operating system software is available from Sun and IBM. This product provides an



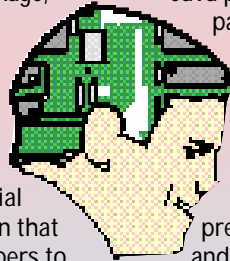
economical way for companies to centrally manage business applications using Java technologies in network computing

MindQ Offers Educational Tutorial

(Herndon, VA) – MindQ Publishing, Inc., a Knowledge Universe Company, has announced that IBM will bundle its tutorial package,

"Introduction to VisualAge for Java and Programming JavaBeans," with IBM's new release of VisualAge for Java 2.0. The tutorial features instruction that will enable developers to get the most out of VisualAge for Java's visual, team-centered, enterprise application development environment. It also includes a full tutorial on creating and using JavaBeans.

MindQ's "Developer Train-



ing for Java" curriculum uses an interactive, multimedia training system designed to address diverse learning styles. It was developed by Java experts to meet the specific needs of beginners as well as the advanced users.

"Essential Java Training" provides what a developer needs to learn the fundamentals of Java programming and prepare for Sun's Java certification. "Advanced Java Topics," designed for developers who want to become experts in Java, contains comprehensive code samples and reference material on topics like JavaBeans and Java APIs.

For more information, call MindQ at 800 646-3008 or visit their Web site at www.mindq.com. IBM's Web site is at www.ibm.com.

environments

The two companies also announced related JavaOS for Business support programs for industry partners, including a full spectrum of software tools, testing facilities and educational assistance, to enable them to build network computing business solutions.

JavaOS software is specifically designed so that companies can centrally store and manage applications used to run their business (such as inventory management or insurance claims

processing) from servers on a network. The servers can be connected to network computers and other thin clients such as kiosks, ticket machines and remote terminals.

JavaOS for Business provides advantages over personal computer operating systems on networks because it enables businesses to manage the entire operating system, services and applications from a centralized server.

For more information, contact IBM at www.ibm.com/java/javaos or Sun at www.sun.com, or contact Datek at www.batavia.com.

Datek Delivers on Sun's JINI Promise

(Kuala Lumpur) – Datek has announced the first Enterprise Software Solution written to Sun's JINI standard for distributed computing. Datek is the

first Java software developer to partner with Sun for its newly launched JINI program. JINI allows computers and devices to have more intelligence in communicating and sharing with other computers across a network, including those that use different operating systems.

Madura, Datek's flagship

ERP and supply chain management (SCM) program, has been released with complete general ledger and systems administration modules. The program emphasizes project accounting, inventory tracking and job costing that allows corporations to know precisely their profit and cost of doing business for any customer or project.

NetObjects Releases NetObjects BeanBuilder 1.0

(Redwood City, CA) – NetObjects, Inc., has announced that it will brand, market and distribute the next version of IBM's subsidiary Lotus Development Corporation's BeanMachine as NetObjects BeanBuilder 1.0. The product enables site builders and Web developers to rapidly assemble and deliver JavaBeans-based Web applications by working in a visual, point-and-click environment, without any programming.

For more information visit www.netobjects.com.

Schlumberger Announces Smart Card

(Austin, TX) – Schlumberger Smart Cards & Terminals has introduced its most powerful member in the Cyberflex family of smart cards. Cyberflex Open 16K doubles the amount of memory available for application software.



Its new features also include a PC/SC interface and fully integrates an application processor, a smart card and a smart card manager.

Schlumberger has the only Web-based smart card support program, with a user discussion forum at www.cyberflex.slb.com.

For more information visit Schlumberger's Cyberflex Web site at www.cyberflex.slb.com or www.slb.com, or call Michele Bernhardt at 408 501-7145.

Datek and Sun are ready to deliver on the notion of the "Internet appliance," making computers and networks as ubiquitous and easy to use as consumer electronics devices.

For more information contact Kamaralzaman Tambu at 603 456-2617 or by e-mail at tambu@pc.jaring.my. Or visit Sun's site at www.sun.com.

Ad

ObjectSpace Announces Voyager Pro 2.0

(Dallas, TX) – ObjectSpace has introduced Voyager Professional Edition, the next generation of widely adopted standards neutral, 100% Pure Java software development platform for distributed object computing.

VoyagerPro combines the power of mobile autonomous agents and remote method invocation with dynamic CORBA and RMI compatibility – providing in a single code base the first-ever seamless support for the most widely used distributed object models.

Developer's can remote-enable Java classes without modification and can write a single line of code to dynamically CORBA-enable Java objects at runtime without modification. The product provides fast and efficient delivery; rich messaging; a multilayered, scalable architecture; and mobile autonomous agents and

Dynamic Aggregation.

For more information, visit ObjectSpace's Web site at www.objectspace.com, or call 972 726-4100. ●

BulletProof Announces JDesignerPro 3.0

(Los Gatos, CA) - BulletProof Corporation introduces a new visual server-side application builder, which includes new features such as the Method Explorer, that facilitate the development of Java modules for server deployment.

The Explorer graphically



shows the breakdown of Java classes. With existing tools developers waste time trying to determine the structure of a Java class file. BulletProof's

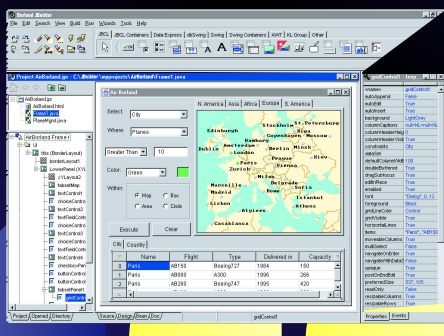
pendent business applications. The high-end version of the product, JBuilder 2 Client/Server Suite, includes support and integration for multiple JDKs, application deployment, Enterprise JavaBeans, Java Servlets, JFC/Swing components, CORBA and high-productivity coding Wizards.

For more information, contact Inprise at www.inprise.com or visit Sun's Web site at www.sun.com. ●

Inprise Joins Sun

(Denver, CO) – Inprise Corporation has entered into an alliance with Sun Microsystems, Inc., to team Inprise's development technologies with the Sun Solaris operating environment. Corporations will be able to take advantage of Inprise's familiar and graphically appealing tools for building and running enterprise applications on the robust and scalable Solaris operating environment.

Inprise also released JBuilder 2 earlier this year. It allows corporations to use the latest 100% Pure Java technologies (including JDK 1.1.6 and JFC/Swing) to rapidly create platform-inde-



Simplicity for Java Introduced

(Summit, NJ) – Data Representations, Inc., has announced version 1.1 of Simplicity for Java. The product is written completely in Java and lets developers build Java applications and applets interactively. Simplicity presents the user with a working model of the actual application that they're creating. All changes to the code are immediately integrated into this working



model without the user needing to save and compile the changes. This dynamic execution reduces the traditional three-step code-compile-test software development process to a single step: design.

For more information, call Carl Sayres at 908 918-0396, fax 908 918-0397, email carl@datarepresentations.com, or visit their Web site at www.datarepresentations.com. ●

new tool allows a simple graphical view, eliminating the need to manually browse through source code to understand which methods called which global variables, accessors and other methods.

JDesignerPro also includes the new SQL Wizard, which allows developers to add SQL statements to code with a few mouse clicks – resulting in flawless syntax.

For more information, visit BulletProof's Web site at www.bulletproof.com. ●

CocoBase Enterprise 2.0 Available from Thought

(San Francisco, CA) – Thought Inc. has announced expanded features of database access framework with CocoBase Enterprise 2.0. To make it even easier to create an application that ties to your database and is high-performance and scalable, the CocoBase Enterprise Framework now includes:

- State-of-the-art object modeling tool Together/J, which can generate all of the database connection code automatically
- Polymorphism support for custom instantiation factories on select() and call()

methods for relational databases

- Automatic configurable shared server-side object caching to optimize and increase performance
- Revised documentation

For more information visit Thought's Web site at www.thoughtinc.com. ●

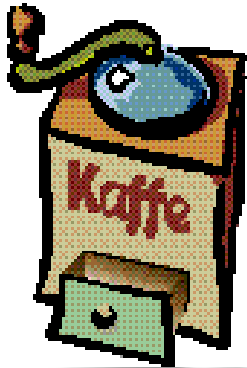
Industry Leaders Bundle Zero G's InstallAnywhere Now!

(Irvine, CA) – ObjectShare will bundle Zero G's InstallAnywhere technology with upcoming releases of PARTS for Java products. In addition, Apple, IBM and Inprise will also bundle the product.

ObjectShare will also extend its current "Delivery Assistant" capabilities to send files to InstallAnywhere providing a total development-to-delivery solution.

InstallAnywhere Now! lists at \$149, and is available free to all registered PARTS for Java users. For more information, visit www.objectshare.com. ●





The Application Server Gold Rush

THE GRIND

by Java George

“The advances in Java technology have made it easy for almost anyone to ‘pan’ for Java’s gold”

It's 1998 and everyone is rushing to get to market and deliver Java-based application servers. The market is heading toward development environments tightly integrated and coupled with application servers. It's like a gold rush to capture the leadership positions with the right solution.

Indeed, the hardware, database and tool vendors are all in the running, throwing their significant weight behind the marketing and development of these products.

Imagine for a moment the short list of the top database vendors: IBM, Oracle, Sybase, Informix, Progress and Microsoft. Only the last member has yet to emerge with or announce plans for a Java-based application server platform. Microsoft, of course, wants nothing to do with Java on the server (the subject of a separate column).

These and other less notable vendors have realized that a tool or database won't present the solution. That might have been the case in good old GUI client/server days, but not with networked, server-centric applications conceived today. In terms of deployment licensing potential, the application server of tomorrow is roughly equivalent to the database server of yesterday.

The database vendors are rushing to provide application servers purposed for Java

Smart customers, or prospects, to be more precise, are demanding: Open Platform Application Server; the ability to CHOOSE the ORB of choice; services such as Transaction Service (JTS) that can be coupled with the AppServer; IIOP protocol so that security layers can be added from third party; adaptive clients ranging from HTML to full-blown Java applications; components on the application server; and RAD environments for development of client/server logic and components.

There's a veritable army of developers (consultants, IS types, VARS), and they've been sweating it out trying to build Internet applications with VI, EMACS and general-purpose tools. They've tried to deploy without application servers and failed. Many have tried to sew disparate client and server processes together and failed once again. They're ready for the promised land!

In effect, the movement of products being sold provides one with a mirror of what the developers are doing (and not doing) with Java and Internet applications.

In early 1997 all we had were lots of cheap Java tools (so-called IDEs). The Apptivity startup was the first on the scene in the spring of 1997 with an integrated development/deployment solution that included a 100% Java application server and professional developer-oriented integrated RAD tool. At that time what was available on the market were inexpensive Java IDEs, the kinds that had three settings: type code, compile code and run code – a sort of Flintstones-type approach to the new breed of three-tier systems development!

Later on, circa winter 1997 (decades later in Web time), we saw a crop of application servers for sale. Most were proprietary extensions of older products but a few new trees represented the above list of desired approaches. Unfortunately, a cart not attached to a horse is of little value. An appserver never truly attached to a comprehensive tool for development presents a serious challenge to the overall equation for success.

And here we are in the middle of the current rush. Some of the products in the market now offer incredible productivity, strong server-side application logic support and the ability to aim for Internet application – and win big. This allows the “mere mortals” out there to become successful within a reasonable time in the deployment of intranets, extranets and Internets. Now, if these mortals could just get past their Y2K problems for a few moments! ☛

Reader Feedback to GUI Client/Server vs Java-Internet/Web Paradigms

Childish Article

I have never seen a more arrogant, uninformed, ignorant, childish article such as this one. You ought to be ashamed of yourself to even publish such garbage.

Frank Calfo

lcandiani@psateam.com

Leading with GUI

Having just read “The Grind” in Vol. 3, Issue 7 I felt the need to get an explanation from the source.

In your article, you contend that there has been little or no

improvement in user efficiency and learning curve with the advent of the GUI. Are you serious? I would suggest that the GUI is responsible for moving the silicon chip from science labs and advanced think tanks to the tune of some 80 million homes in America.

If folks still had to contend with cryptic codes, isolated applications, lack of drag and drop, etc., most of the Java developer community would be out of a job. It's the GUI which led to such a boom in the Internet.

While I grant that many of the advances have been made as a by-product of the GUI, there's no question that GUI led the way. I'll agree with your assertion that early in GUI development the tools and methodologies were cumbersome and expensive, but not any more.

Today an entry level programmer can create an intuitive and familiar interface without a single line of code. While it may not be an enterprise application, it's a first step.

Todd Freeman

tfreeman@filenet.com

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



George@sys-con.com

Ad

Full Page Ad