// JANUARY/FEBRUARY 2012 /

Java™ magazine

By and for the Java community

10
JAVA VS. SPAM
Mollom eliminates unwanted postings from Websites worldwide

33
PROJECT COIN IS A TIME-SAVER
How Java SE 7's language changes streamline your code

68
FIX THIS
Test your knowledge with a Swing conundrum

TAME THE CLOUD

How Java EE is evolving to become the cloud computing platform standard

ORACLE.COM/JAVAMAGAZINE

ORACLE

COVER ART BY I-HUA CHEN
PHOTOGRAPH BY DAVE BRADLEY

Sometimes, buzzwords are for real. *Cloud computing* is one such example: most will agree that the term itself is overused and lacks a clear, universally accepted definition. But we can also agree on the importance of understanding cloud computing's likely impact on the application development process. Pretending otherwise would constitute a serious breach of career-management duty.

In some quarters, that impact is expressed in the context of the emerging "DevOps" principle, which calls for developers to use their programming powers to free themselves from the burden of IT operations. Cloud computing, by almost any definition, is a key DevOps enabler.

Currently, the Java EE platform lacks the abstractions that allow Java developers to move confidently in that direction. But with Java EE 7 (JSR-342; final release expected later in 2012), that will change—with the inclusion of updated APIs, roles, and more that will bring support for cloud service requirements such as multitenancy, elasticity, and scalability directly into the standard. As a result, apps written to target Java EE 7 app servers (which will in fact become services themselves) will be much more natively cloud-aware than they are today. What could be more DevOps-friendly than that?

Our interview with Cameron Purdy, Oracle vice president of Java EE development, reveals the reasoning behind these efforts in more detail. (You can also review the Java EE 7 JSRs.) And as usual, you'll find this issue packed with informative, actionable articles on other subjects as well—including wrap-ups of JavaOne Latin America and Devoxx 2011, a Swing JLayer primer by Josh Marinacci, an introduction to Java SE 7's Project Coin features by Julien Ponge, and more. Enjoy it!

Justin Kestelyn, Editor in Chief BIO

//send us your feedback /

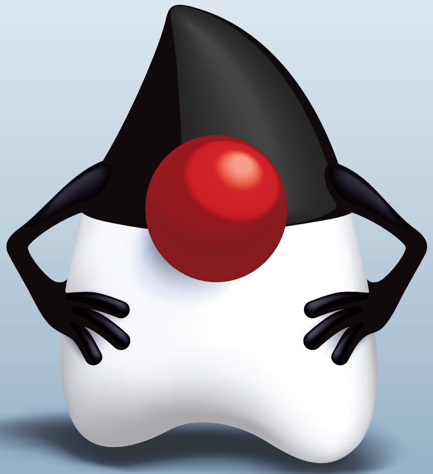
We'll review all suggestions for future improvements. Depending on volume, some messages may not get a direct reply.

PHOTOGRAPH BY BOB ADLER

Java™

GIVE BACK!
ADOPT A JSR

Find your JSR here



/n software

We didn't invent the Internet...

...but our components help *you* power the apps that bring it to business.

PURE JAVA COMPONENTS & ENTERPRISE ADAPTERS FOR

- **E-Business**
  AS2, EDI/X12, NAESB, OFTP …
- **Credit Card Processing**
  Authorize.Net, TSYS, FDMS …
- **Shipping & Tracking**
  FedEx, UPS, USPS …
- **Accounting & Banking**
  QuickBooks, OFX …
- **Internet Business**
  Amazon, eBay, PayPal …
- **Internet Protocols**
  FTP, SMTP, IMAP, POP, WebDav …
- **Secure Connectivity**
  SSH, SFTP, SSL , Certificates …
- **Secure Email**
  S/MIME, OpenPGP …
- **Network Management**
  SNMP, MIB, LDAP, Monitoring …
- **Compression & Encryption**
  Zip, Gzip, Jar, AES …

The Market Leader in Internet Communications, Security, & E-Business Components

Each day, as you click around the Web or use any connected application, chances are that directly or indirectly some bits are flowing through applications that use our components, on a server, on a device, or right on your desktop. It's your code and our code working together to move data, information, and business. We give you the most robust suite of components for adding Internet Communications, Security, and E-Business Connectivity to any application, on any platform, anywhere, and you do the rest. Since 1994, we have had one goal: to provide the very best connectivity solutions for our professional developer customers. With more than 100,000 developers worldwide using our software and millions of installations in almost every Fortune 500 and Global 2000 company, our business is to connect business, one application at a time.

To learn more please visit our website → **www.nsoftware.com**

Georges Saab

NEWS ROUNDUP

JavaOne LATIN AMERICA 2011

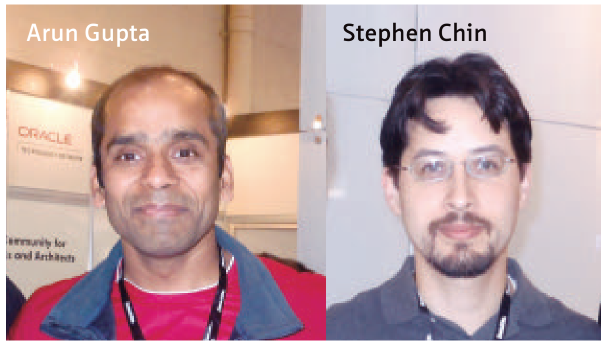
JavaOne Latin America 2011 was held in São Paulo, Brazil, December 6–8, 2011. The regional JavaOne conference included great keynotes, deep technical sessions, engaging hands-on labs, a bustling Oracle Technology Network lounge, booth swag (Duke T-shirts!), wonderful conversations, and even a maté tasting.

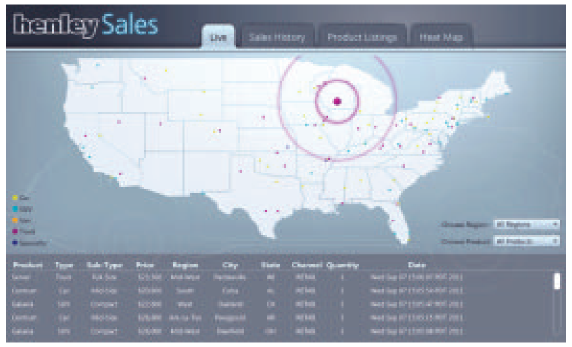
Nandini Ramani

PHOTOGRAPHY BY LOAINE GRONER AND ARUN GUPTA

PROMISES DELIVERED

At the Java strategy keynote on the first day of JavaOne Latin America, Oracle executives stepped through Oracle's Java technology roadmap for Java SE, Java EE, Java ME, and JavaFX. "Last year at JavaOne Latin America, Oracle was full of promises," said **Georges Saab**, vice president of development, Java Platform Group. "This year we can say Oracle delivered on those promises: Java SE 7, JavaFX 2.0, and increasing transparency in the JCP [Java Community Process]." **Nandini Ramani**, vice president of development, Java client platform, announced the release of JavaFX 2.0.2. She explained that starting with JavaFX 2.0.2, JavaFX is available under the same license and business model as Java SE. This includes the ability to distribute the JavaFX runtime (or Software Development Kit) for third-party developers with their application(s), subject to the terms and conditions of the license. (Developers can follow and join the JavaFX open source project at OpenJFX.) The technical keynotes on the second day provided more details on each of these technologies.

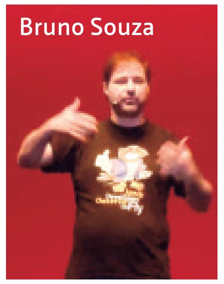

Arun Gupta
Stephen Chin



The source code for the Henley Sales Dashboard is available as part of the JavaFX Sample Showcase (currently only for Windows).

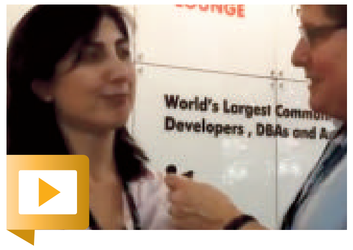
FOCUS ON JAVAFX

There was lots of buzz around JavaFX. At the technical keynote, the JavaFX demos generated the most interest, especially the JavaFX game running on a desktop and two different tablets. The other demo that sparked a lot of interest was the Henley Sales Dashboard, a client/server application for a fictional global automobile company. The front end was developed with JavaFX, displaying real-time sales data via MySQL and deployed on Oracle GlassFish Server. At the standing-room-only session "XML-Free Programming: Java Server and Client," **Stephen Chin**, chief agile methodologist at GXS and Java Champion, and **Arun Gupta**, Java evangelist at Oracle, discussed how to build Web apps with a JavaFX client and a Java EE server.



BEST FOR LAST: THE COMMUNITY KEYNOTE

The JavaOne Community Keynote was informative and funny. **Fabiane Nardon** and **Bruno Souza** of Brazilian Java user group SouJava opened the keynote with their list of the top five developments in Java for the year: 5) Java on Mac to OpenJDK, 4) SouJava and the London Java Community getting seats on the JCP Executive Committee, 3) Twitter adopting Java for improved performance, 2) Java SE 7 being released, and 1) JavaFX being made open source. Nardon then discussed women in technology and, with SouJava members **Yara Senger**, **Ana Abrantes**, and **Loiane Groner**, announced the formation of jDuchessBR, a group for Brazilian women using Java. Then **Vinicius Senger** gave a demo of the Duke's Choice Award winner, jHome, showing that what you can do with Java is limited only by your imagination: adjust the lights in your house via Twitter, brew coffee, or monitor someone's heart rate remotely (Arun Gupta was the guinea pig). The keynote concluded with a twist on the *Java Life* video entitled *Real Java Geeks*. See what Java developers do when they aren't in their cubicles (or offices, home offices, or coffee shops).



Fabiane Nardon talks about zero downtime and the amazing performance of Java EE.

When they're not coding hard, Java developers surf, hike, ride motorcycles, fish, practice jujitsu, and so much more.

THE COMMUNITY THAT RIDES TOGETHER . . .

In true community spirit, **Fabiane Nardon** and other SouJava members put together a Java community bike ride the Sunday morning before JavaOne Latin America. Nardon arranged for bike rentals and matching jerseys, and "bike angels" made sure no one was left behind. Nearly 30 riders enjoyed slightly overcast and cool weather and a wonderful route through São Paulo, thanks to Sunday lane closures just for bicyclists. The riders looked great in their fabulous Duke bike jerseys. The route was 12 miles (22 kilometers)—check out the GPS documentation and full redundancy. The group rode at a leisurely pace that allowed everyone to chat and meet new friends. The group even visited two parks and a street fair along the way. Affectionately called the Geek Bike Ride, it had just the right combination of exercise, technology, and fun. Watch for #geekbikeride to join or start a community bike ride near you.

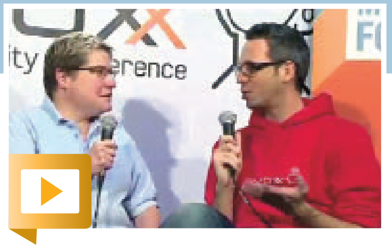
PHOTOGRAPHY BY ARUN GUPTA AND TORI WIELDT

Parleys.com Brings Sessions to You

Couldn't make it to JavaOne or Devoxx? That needn't stop you from "attending" many of the most important conference sessions. You can do so at Parleys, the GlassFish-powered e-learning platform invented by Java Champion, Belgian Java User Group leader, and Devoxx Founder Stephan Janssen. The Parleys team recorded 40 percent of the JavaOne 2011 sessions, along with a substantial sampling of Devoxx 2011 sessions and interviews.

A Parleys technical session presentation provides you with a full educational experience, including audio, slides, and video of demos from the sessions. You hear and see almost everything the session's live attendees heard and saw—with the added benefit that you can pause, back up, and review a session segment or slide any time you like. You can watch Parleys presentations online, or you can download them for offline viewing using the free Parleys Desktop application.

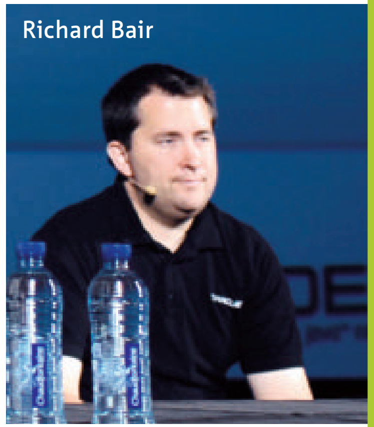
Visit the Parleys JavaOne channel to view popular sessions from JavaOne 2011 including "Introduction to JavaFX 2.0," "Java EE 6: The Cool Parts," and "The Heads and Tails of Project Coin."


Stephan Janssen talks about the challenges of putting on Devoxx.

Brian Goetz, Mark Reinhold, Ben Evans

Joe Darcy

Michael Heinrichs


Richard Bair

Devoxx 2011: TECHNICAL IMMERSION

Devoxx 2011 celebrated its 10-year anniversary in Antwerp, Belgium, in November 2011. Sold out every year weeks in advance, Devoxx has become the biggest Java developer conference in Europe, with 3,350 attendees from 40 countries and more than 170 well-known speakers giving 150 sessions over five days.

The conference is a technical immersion. Junior and seasoned developers learn about platform improvements and better and faster application developments with a mix of keynotes, technical overviews, three-hour sessions, hands-on labs, 30-minute "Tools in Action" talks, and the traditional birds-of-a-feather discussions.

Henrik Stahl and **Cameron Purdy** of Oracle talked about Oracle's investment in and innovation around the Java platform with the upcoming releases of Java SE 8, Java EE 7, and JavaFX 3.0. **Mark Reinhold**, **Joe Darcy**, and **Brian Goetz**, Java SE architects and technical leads at Oracle, explained the latest progress with Project Coin, Project Jigsaw (modularity), and Project Lambda in a series of sessions and casual discussions with developers. Java EE 7 development will be multitenant, on demand, and autoprovisioned for the cloud platform. **Jerome Dochez**, a GlassFish architect at Oracle, discussed cloud infrastructure and platform-as-a-service enhancements in Java EE 7 and GlassFish Server Open Source Edition 4. Spec leads presented specifics on JSRs: Oracle's **Marek Potociar** on JAX-RS 2.0 and the upcoming and sought-after client API; Oracle's **Shaun Smith** on JPA 2.1; Oracle's **Nigel Deakin** on Java Message Service 2.0; and Ehcache Founder **Greg Luck** on javax.cache.

Shaun Smith

JavaFX 2.0, released at JavaOne, had a big presence too. Oracle technical leads **Jasper Potts**, **Richard Bair**, **Michael Heinrichs**, and **John Yoong** and two community leaders, **Peter Pilgrim** and **Stephen Chin**, presented seven talks for a full immersion on features, deep dives about animations and custom bindings, and an open mic session. The "Why We Should Target Women" panel attracted a full house as well. Only 1 percent of Devoxx attendees are women, so the discussion centered on encouraging women into the field and the conference. Here are a few of the ideas discussed:

1. Parents should encourage their daughters to play with Legos and learn programming.
2. More organizations should target young women in high school and college to expose them to programming. Women need mentors (men or women) to learn to become speakers at conferences and to promote themselves better.

Devoxx Heads to France

To celebrate its 10-year success, Devoxx will expand to France, in the first conference modeled after Devoxx in Belgium. At Devoxx France, April 18–20, 2012, 75 percent of the sessions will be in French and 25 percent in English. The Paris Java user group behind this event promises a Devoxx experience plus spring in Paris and, of course, wine. A call for papers is open, and Oracle is a sponsor.

It's all happening in Paris.

PHOTOGRAPHS COURTESY OF DEVOXX

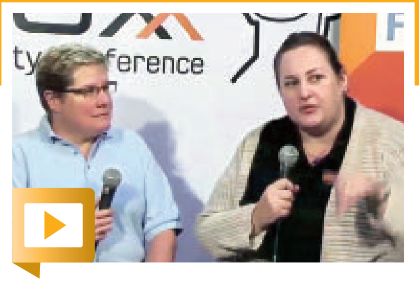
WHAT'S NEW AT JAVA.NET

AFTER A YEAR OF UPHEAVAL AND REBUILDING, JAVA.NET IS GROWING AGAIN, WITH 80 TO 90 NEW PROJECT REQUESTS PER WEEK, GROWTH IN MEMBERSHIP, AND NEW COMMUNITY VOLUNTEERS TAKING OVER LEADERSHIP ROLES IN SEVERAL COMMUNITIES. Plans for 2012 include new hardware and performance improvements, adding another level of social and group functionality (starting with Java user groups and then the Java Community Process), a complete redesign in how communities are structured, and improvements to the project catalog and search facilities. **Sonya Barry**, Java.net community manager, says she has heard many great ideas from community members that she will be looking into, ranging from opening the API so users can create mobile apps that work with the site and improving search engine optimization on projects and people to using geotags to let users know where their nearest Java user group is.

Java.net blog posts on a variety of topics are receiving considerable attention, and innovative new Java-centric open source projects are underway (and seeking added participation from the developer community as well).

Recent Java.net blogs of note include **Osvaldo Pinali Doederlein**'s JavaFX Balls 3.0, **Mamadou Lamine Ba**'s What JSF Should Become?, **Santiago Pericas-Geertsen**'s JAX-RS 2.0—Client API, **Cay Horstmann**'s Operator Overloading Considered Challenging, and **Tushar Joshi**'s Concentrating on Task in Hand (Similar to Mylyn) in NetBeans IDE.

Innovative new Java.net projects include JSF extension points, an easy-to-use, powerful drop-in/plug-in system for Java EE; SWELL, an English-like domain-specific language for Swing testing; and the Vorpal XMPP Component Framework, an injection-based XMPP component framework for Jabberwocky. More than 2,200 diverse Java.net open source projects are available. They welcome your participation.


Tori Wieldt talks with Sonya Barry about the Java.net migration.

PHOTOGRAPH BY CAITLIN YOUNGQUIST

Adopt-a-JSR

Want to help move Java forward? Is there some specific enhancement to the Java platform that would benefit you and your industry peers? Your Java user group (JUG) can make that happen by participating in the Adopt-a-JSR program. To contribute to a JSR, your JUG will have to join the Java Community Process (JCP)—a procedure that is being streamlined for JUGs as part of the JCP's new openness. JUGs have already contributed significantly to changing the JCP through their involvement in JCP.next.

As London Java Community leader (and JCP member) **Martijn Verburg** (@Karianna) notes, "there are genuine domain experts in JUGs that can serve as EG [Expert Group] members; skilled developers that can help with RI [reference implementations] and TCK [technology compatibility kit] implementations; and enthusiastic hordes that can help test early versions, give feedback, and help deal with project overheads such as mailing lists and issue trackers as well as promoting the JSR in general."

Martijn Verburg explains the Adopt-a-JSR program.

Twitter and Azul Elected to the JCP

Twitter and Azul Systems were elected to the Java Community Process (JCP) Standard/Enterprise Edition Executive Committee (EC) in the JCP's fall 2011 election. The companies bring widely recognized expertise in Java Virtual Machine (JVM) performance to the JCP at a critical time in the history of Java's evolution.

In April 2010, Twitter engineers announced that they had achieved a three-times performance improvement in Twitter search, in part by replacing their Ruby on Rails stack with a new Java service built on the JBoss Community's highly scalable NIO client/server socket framework, Netty. At JavaOne, Twitter's **Attila Szegedi** spoke about the team's accomplishment at the session "Everything I Ever Learned About JVM Performance Tuning at Twitter." Azul develops high-performance Java products, such as the Linux-optimized Zing JVM and the C4 Pauseless Garbage Collector. The companies were elected to the two open election EC seats out of an unusually large field of nine candidates.

NetBeans IDE 7.1 Released

NetBeans IDE 7.1, launched in January 2012, is the first integrated development environment (IDE) to support JavaFX 2.0. Developers can create JavaFX applications completely in the Java programming language using industry-leading development support provided by NetBeans. Also available is support for creating JavaFX FXML–based applications or traditional Java-based JavaFX applications, and JavaFX Preloader applications to customize the startup experience of a JavaFX application. Developers can also package their applications as a standalone application, a Java Web Start–based application, or an in-browser application. Debugging the JavaFX application UI is now simplified with the new GUI visual debugger, which can also be used with Swing-based UIs.

This release introduces batch refactoring tools that enable projectwide refactorings, including an extensible "inspect and refactor" tool, new refactorings and code hints, a block selection mode, and an import statement organizer, among many other Java editor enhancements. NetBeans IDE 7.1 supports Oracle WebLogic Server 12*c*, providing industry-leading Java EE 6 support to developers using the Oracle WebLogic platform.

EVENTS

MOBILE WORLD CONGRESS *FEBRUARY 27–MARCH 1, BARCELONA, SPAIN*

"Redefining mobile" is the theme of this year's mobile technology and business conference. More than 60,000 attendees are expected for this event, which includes a conference program, exhibition, and awards program. App Planet, a conference in its own right, is the world's largest gathering of apps developers—more than 10,000 are expected.

FEBRUARY

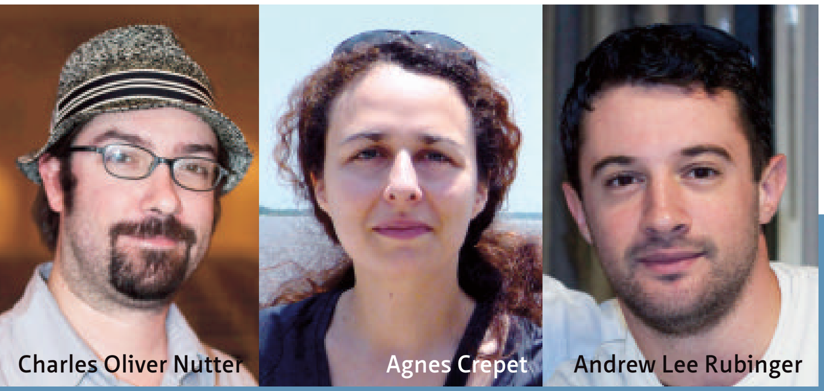
Java Performance Tuning Workshop
FEBRUARY 6–9, SYDNEY, AUSTRALIA
This four-day course will be led by **Kirk Pepperdine**, Java performance tuning expert, Java Champion, and international speaker and trainer. Each day will have a topic: introduction to performance, the Java Virtual Machine, tooling, and performance tuning in practice.

Jfokus 2012
FEBRUARY 13–15, STOCKHOLM, SWEDEN
Jfokus is the largest annual conference for anyone who works with Java in Sweden. The conference is arranged together with Javaforum Stockholm, a Swedish developer community and Java user group. With speakers from Sweden and around the world, the conference focuses on development with Java and surrounding techniques.

MARCH

QCon London 2012
TRAINING MARCH 5–6, CONFERENCE MARCH 7–9, LONDON, ENGLAND
QCon London is the sixth annual London enterprise software development conference designed for developers, team leads, architects, and project managers. Day two of the development track focuses on hard-core Java.

ILLUSTRATION BY I-HUA CHEN, PHOTOGRAPH BY RON SELLERS



Heroes of Java

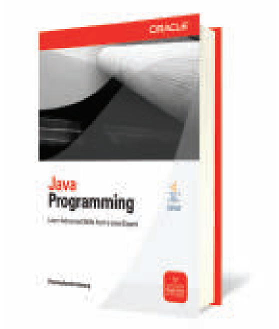
There are many heroes of Java out there—people who are making significant contributions on a daily basis within the Java ecosystem, thereby building Java's future in the most practical way possible. Often their work is performed far away from the public spotlight. Software architect and author Markus Eisele is highlighting these people and their efforts in his new interview series, *The Heroes of Java*.

The series is not about celebrities; rather, the focus is on "people displaying courage and sometimes even self-sacrifice for the greater good," says Eisele. Heroes in the series include Marcus Lagergren, Oracle JRockit architect and team lead; Andrew Lee Rubinger, JBoss software engineer; Cay Horstmann, author and professor; Charles Oliver Nutter, JRuby developer; and Java user group leaders Martijn Verburg and Agnes Crepet.

If you'd like to suggest someone worthy of inclusion in the series, Eisele invites you to post a comment to his lead *Heroes of Java* entry.

PHOTOGRAPHS COURTESY OF MARKUS EISELE

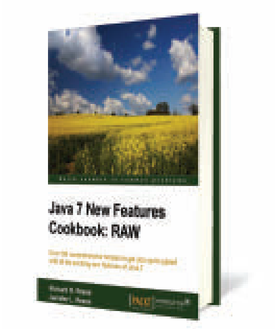
JAVA BOOKS



JAVA PROGRAMMING

By Poornachandra Sarang
Oracle Press (January 2012)

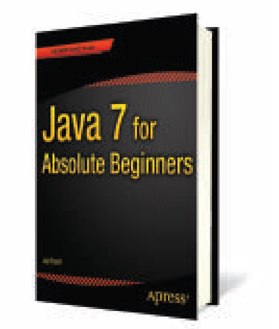
Java Programming features hands-on programming exercises and examples based on the author's 10-plus years of teaching Java to experienced professionals. This Oracle Press guide covers all the new Java features, including multilingual support, support for JavaScript and other scripting languages, JavaFX, and several other modifications to the language. Java programmers will find exactly what they need to know in order to integrate these new features into their skill set. The author also provides thorough coverage of Java syntax, the event model, threads, network programming, I/O programming, and applets.



JAVA 7 NEW FEATURES COOKBOOK

By Richard M. Reese and Jennifer L. Reese
Packt (May 2012)

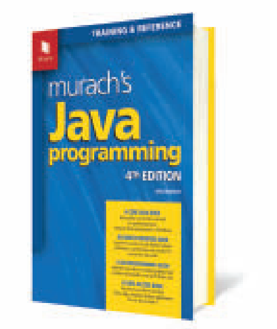
Java 7 New Features Cookbook offers a practical, recipe-based approach for learning about the new features of Java 7. The book starts with coverage of new language improvements. Subsequent chapters address new features of Java 7 while incorporating these new language improvements when possible. Content includes NIO techniques, GUI improvements including window methods, the new JLayer class, various dialog-box-related methods, and much more. The book is currently available as a RAW book (read as it is written).



JAVA 7 FOR ABSOLUTE BEGINNERS

By Jay Bryant
Apress (December 2011)

Java 7 for Absolute Beginners introduces the new core open source Java Development Kit. The book focuses on practical knowledge, providing all the bits and pieces an utter novice needs to get started programming in Java. The book teaches Java development in a language that anyone can understand. It provides simple, step-by-step examples that make learning easy, allowing you to pick up the concepts. It also offers clear code descriptions and layout so that you can get your code running as soon as possible.



MURACH'S JAVA PROGRAMMING, 4TH EDITION

By Joel Murach
Mike Murach & Associates (November 2011)

Get a fast start in Java, and then master all the core skills you need to be a professional Java developer. By the end of Chapter 6, you'll be writing fail-safe, object-oriented applications with business classes and objects. By the end of the book, you'll be working comfortably with features such as inheritance, interfaces, arrays, collections, threads, GUIs, files, and databases (using JDBC). At every step, you'll save time by using NetBeans. Read a sample chapter.

Winning the War Against Spam WITH JAVA

Mollom eliminates unwanted postings from Websites worldwide.

BY DAVID BAUM AND ED BAUM

Got spam? Who doesn't! But if you think finding and deleting a few hundred spam messages from your inbox or Website is a challenge, try deleting a few hundred million. That's the volume that leading spam-fighters such as Dries Buytaert battle each day. As cocreator of Mollom, a Java-based technology that processes hundreds of requests per second to keep the invading horde at bay, Buytaert is continually confronted with the "dark side" of the internet: it has become a petri dish of sorts, breeding countless unsolicited automated messages and comments like invasive bacteria. This infestation weakens worldwide productivity and threatens to compromise the way the Web works. The battle for the internet is raging, and it's us against the spambots.

These *spambots*—automated programs that assist in sending and posting spam—are getting smarter and more intrusive all the time. Thankfully, defensive strategies keep getting better too, as evidenced by Buytaert and his team. They recently migrated Mollom from home-

Dries Buytaert, Mollom cocreator, at the company's Burlington, Massachusetts, office

PHOTOGRAPHY BY DAVE BRADLEY

SNAPSHOT

MOLLOM
mollom.com

Headquarters:
Antwerp, Belgium

Industry:
Open source software

Employees:
4

Java version used:
Java EE 6

Buytaert drops in and answers questions from participants in a class at Mollom's Burlington, Massachusetts, office.

grown systems to GlassFish Server Open Source Edition—the open source version of Oracle GlassFish Server—to better manage the demands of keeping some of the world's busiest Websites free from spam.

"Java is a great fit for what we do," Buytaert says. "Because Java is part of a much larger technology ecosystem that includes lots of development tools and open source application servers like GlassFish Server Open Source Edition, it enables us to create a hosting service for thousands of customers with the back-end infrastructure that delivers the reliability, scalability, and availability of an enterprise-scale information system."

BEYOND PHISHING

Internet spam and the tactics used by spammers have become increasingly sophisticated since first appearing on internet bulletin board systems (BBSs) in the 1980s. Back then, spamming was as simple as repeating a word or phrase over and over to scroll other users' text off the screen. By the early 1990s, spam had evolved, and multiple repeat postings began to flood Usenet, where the first commercial spam messages appeared.

Today, most of us know spam as an abuse of our e-mail inboxes. And even though only a small percentage of people purchase the products (or fall prey to the scams) advertised through spam, these e-mail messages continue to proliferate because of their extremely low cost. If you manage a Website, you know how complex spamming strategies have become. Some spam messages on blogs and forums are targeted toward readers in an attempt at product marketing or *phishing* (trying to acquire information such as usernames, passwords, and credit card details). However, the spam that Website administrators battle most is a tactic deployed in an attempt to increase page rank on search engines such as Google.

"The more incoming links you have to your sites, the more relevance Google will give to those sites," explains Buytaert. "The spammers' goal is to abuse Websites' commenting systems to create links back to their own Websites in an effort to get the number one spot for certain keyword searches."

Buytaert says the strategies these spammers use are becoming so clever that it's often difficult to label the invasive content as spam. "They provide contextually relevant comments along with links back to their sites," he says. "The line between what's spam and what's not gets blurry. Identifying and blocking these types of abuses is becoming very difficult for Website owners and administrators."

ACCURATE AIM

Mollom's efficiency rate at identifying spam is 99.96 percent. Only 4 messages out of 10,000 succeed in fooling its Java programs.

The challenge of blocking this type of spam gets even greater when you consider its volume. Since Mollom's launch in 2008, the service has blocked more than half a billion spam messages. In fact 82 percent of all the comments on the sites Mollom monitors are spam.

Buytaert is a prominent figure in the Java community. In 2008 when he was at Belgium's Ghent

Buytaert and Mollom team members break for a quick lunch.

University defending his PhD thesis about Java performance and improvements, his defense committee included the father of the Java programming language, James Gosling, who was then senior vice president at Sun Microsystems, and Michael Hind, a senior IBM staff member for Java research.

Buytaert is best known as the founder and project lead of the Drupal project, an open source content management platform that powers about 2 percent of all the Websites on the internet. He has been using Drupal for his personal blog for many years, and as his blog became more popular over the years, spam comments proliferated. "I was wasting time manually removing unwanted content from the site," Buytaert says, "and I noticed a lot of other people in the Drupal community with the same problem. I saw an opportunity not only to improve my situation, but also to give back to the community. That led me to start working on Mollom."

TYPE THESE LETTERS, THEN DO A BACKFLIP

As kids, most of us probably never dreamed that we would someday be asked to prove to a computer that we're human. Yet that's what the ubiquitous CAPTCHA antispam solutions accomplish with their tedious challenge/response tests. The distorted backgrounds and warped text provide a problem easy enough for most humans to solve, while impeding most automated software. We dutifully squint at the twisted letters and numbers and retype them. Not only is it a nuisance to Website visitors; it's also discriminatory against visually impaired people. And some spambots are becoming smart enough to solve the CAPTCHA challenges by using advanced optical character recognition techniques.

Mollom's solution to spam is unique in that it intelligently combines three of the most effective defenses against unwanted automated comments while remaining mostly invisible to legitimate site visitors and comment posters. Mollom only delivers a CAPTCHA challenge as a last resort—for most cases, a CAPTCHA is never shown, which makes it easier for people to visit Mollom-protected Websites. Mollom achieves this feat by deploying other antispam techniques first. For example, it uses a clever trick based on JavaScript to hide form fields from human view. People don't see the form fields, but spambots assume the form fields are visible so they fill them out. When they do, their submissions are blocked. This technique is known as a *honeypot*.

Text analysis is another element of the Mollom solution. Like Defensio or Akismet (the latter being the default spam filter in WordPress blogs), Mollom examines the written content of a post to establish whether it's a legitimate submission or spam. The problem is, sometimes it's difficult to determine. When used alone, text analysis can wrongfully classify a legitimate post as spam, and block a genuine user. "Mollom recognizes that some posts can be difficult to categorize," Buytaert explains. "It's not always black or white, spam or not spam. So we have a third category: *unsure*. When Mollom is unsure if a post is spam or not, it serves a CAPTCHA."

Because it's a hosted software-as-a-service application, Mollom benefits from network effects. All the people using the system work together, providing feedback that helps the spam-identifying classifiers improve. In other words, Mollom continually gets smarter. "Using these network effects, we can even observe spambots as they assault multiple sites simultaneously,"

JAVA ON THE FRONT LINES

Java-based Mollom has blocked more than half a billion spam messages. Approximately 82 percent of comments on the sites Mollom monitors are spam.

Mollom: Under the Hood

The initial version of the back end for Mollom, a Web service that keeps invading hordes of spambots at bay, was based on a custom Java SE–based XML-RPC server. To allow increased focus on Mollom's core Machine Learning technology, Mollom co-creators Dries Buytaert and Benjamin Schrauwen decided to switch to a Java EE 6 back end, hosted on a GlassFish Server Open Source Edition 3.1.1 cluster. The Mollom team now extensively uses several core technologies such as JAX-RS with Jersey, the Java Persistence API, and Enterprise JavaBeans 3.1 (both singleton and stateless session beans). This has enabled the team to quickly develop an extended REST version of their API. For storage, Buytaert and Schrauwen chose a Cassandra cluster, because the back end is very write-intensive. Cassandra also allows the Mollom team to easily scale the storage layer across multiple data centers. Currently Mollom is running on dedicated machines, and to further reduce maintenance work, Buytaert and Schrauwen plan to move the infrastructure to Amazon's cloud environment, and to take advantage of the upcoming cloud features in GlassFish Server Open Source Edition.

Buytaert says. "We know that's not human behavior. It helps us identify and block these malicious automated programs."

By intelligently combining all these techniques and only using the CAPTCHA as a final test to weed out the small number of false positives and false negatives, Mollom provides the best-possible quality in terms of blocking spam while delivering the best-possible user experience for visitors to the sites it protects.

ENLISTING JAVA

Mollom is a real-time service, eliminating spam, filtering profanity, and safeguarding all kinds of sites—from small blogs that may only get one spam message per week to large social networks such as Europe's Netlog, with its 85 million registered users, and large media companies such as Sony Music Entertainment, which host hundreds of popular artists' sites. When someone makes a comment on a protected site, that site sends the comment to Mollom. Mollom analyzes the content and then returns a reply to the site. "It's important for us to have a low latency so we can process complex statistical analysis in milliseconds," Buytaert explains. "Visitors to Websites don't want to wait after clicking Submit to see if their comments were accepted. To accomplish this, we need a technology that can keep a lot of data in persistent memory so we can instantly classify content."

Buytaert clears his head by Mollom's "campfire," a place where staffers can brainstorm or just relax.

When Buytaert created Mollom with his partner, Benjamin Schrauwen, they initially built everything from scratch. But as Mollom grew to the point where it was handling millions of messages per day, Buytaert and Schrauwen found that they were spending too much time on infrastructure-related issues. That's when they switched to GlassFish Server Open Source Edition.

"We migrated to GlassFish because it let us worry less about memory management, XML parsing, database connection pooling, REST handling, and a lot of other factors that make big information systems work," Buytaert says. "It lets us scale that runtime environment more effectively. By migrating from our home-grown Java-based solution to GlassFish, we freed ourselves to focus more on the domain-specific challenges of Mollom. It's a great fit."

Buytaert says new users can get Mollom up and running within minutes. They simply create an account on mollom.com, install a plug-in, and enter public and private keys.

"The volume of spam comments we block is one measure of Mollom's success, but the quality of our service is equally important," Buytaert says. "Our efficiency rate is currently 99.96 percent. That means if we see 10,000 messages, we make four mistakes. We know we are helping to keep the Web a little bit cleaner. In business terms, that means Website owners can spend a lot less time worrying about the content that doesn't belong and more time creating valuable services for their user base. </article>

David Baum and Ed Baum are freelance writers specializing in innovative businesses, emerging technologies, and compelling lifestyles.

THE ENGINE OF CHOICE

A new rate and availability information service, powered by Java, puts **Choice Hotels** front and center on Google Maps. **BY PHILIP J. GILL**

Rain Fletcher (left), vice president, application development and architecture, Choice Hotels International, updates Todd Davis, chief technology officer, on current projects.

PHOTOGRAPHY BY PHIL SALTONSTALL

Thanks to the internet, the business model and distribution channels that sustained the travel and hospitality industry for more than two decades have been transformed. The latest generation of mobile technologies only intensifies the competition for internet traffic and reservations between hoteliers, traditional distribution systems, and a plethora of online travel agencies, including Expedia, Travelocity, and Orbitz. In such an intensely competitive environment, any new technology or service that brings more eyeballs to a hotel chain's site, bypassing distribution partners as well as rivals, can offer a significant opportunity to gain competitive advantage and add revenue, says Rain Fletcher, vice president, application development and architecture, at Choice Hotels International.

"Our goal is to drive as many of our reservations onto our Website or into our call center as possible," explains Fletcher.

A typical day for Fletcher: back-to-back meetings with staff and vendors; a quick phone call while grabbing coffee.

SNAPSHOT

CHOICE HOTELS INTERNATIONAL
choicehotels.com

Headquarters:
Silver Spring, Maryland

Industry:
Hospitality

Revenue:
US$642 million in FY 2010

Employees:
1,800 (U.S. only)

Java version used:
Java Development Kit 6

That's why Choice Hotels, one of the world's largest lodging chains, was intrigued by the prospect of adding Google to its portfolio of distribution partners. The Web search giant asked Choice to participate in a new information service that it was developing for its popular Google Maps and Google Places services. The new service, launched in July 2011, displays real-time hotel room availability and rates alongside location maps.

"If you go to maps.google.com or even just google.com and search for hotels in Phoenix or Los Angeles, for instance, you'll get a map with pinpoints with letters—*A*, *B*, *C*, *D*—next to them," explains Fletcher. "The letters correspond to a hotel that's on the nav [navigation] bar at left. The nav bar displays information about that hotel, including the name, location, and star rating. Then there's a little drop-down box that shows you the hotel and the rate for that hotel from a number of different sources. If you click on the link for Choice Hotels, it takes you to our Website to book that specific hotel."

That link is a "deep" click-through that brings potential customers directly to Choice's own Website—where, it is hoped, they will book a room. According to Fletcher, Java was uniquely suited as the language and platform to create this unique, innovative service.

"Java is a true enterprise-class development platform," says Fletcher. "We operate at internet scale. We do US$7 billion in revenue a year through our central channels, so we definitely have enterprise-class problems to solve. Java is uniquely positioned as a development platform to solve those."

EARLY DAYS
Choice started out as a loose affiliation of seven independent motel owners in Florida in the 1930s.

THE CORE OF CHOICE

Founded in 1939 and based in Silver Spring, Maryland, with IT operations in Phoenix, Arizona, Choice Hotels is one of the leading lodging companies in the world. Structured on the franchise model, the chain has more than 6,000 hotels in the U.S. and in more than 30 countries around the globe. What began as a chain of motels has blossomed into Choice's 11 brands, which range from budget Econo Lodge and Rodeway Inns and

Fletcher catches up with his staff during a hallway mini-meeting; a Choice staffer crushes his ping-pong opponent (an Oracle rep) while Fletcher looks on.

midrange Quality Inns and Comfort Suites to full-service Cambria Suites and the boutique and historical hotels that are part of the Ascend Collection.

Like others in the lodging industry, Choice's IT infrastructure is a hodgepodge of decades-old legacy systems, some based on the first computerized reservation system—originally designed for the airline industry 25 years ago—mixed in with newer systems based on proprietary languages and protocols and some more-recent technologies, such as C, Java, and even a little .NET. Its core is its central reservation system (CRS), which stores up-to-date information about Choice's inventory of rooms, room rates, and availability. Layered on top are applications and processes that push that information out to its Website, reservation call center, and third-party distributors.

Twenty years ago, these distributors included global distribution systems (GDSs), such as Sabre, Worldspan, and Galileo, and members of the Open Travel Alliance (OTA). When the travel industry began to establish a presence online, Choice added online travel bureaus such as Expedia, Travelocity, and Orbitz to the mix as well. While Choice cooperates with these partners, it also competes with them, notes Fletcher.

"The ecosystem or the set of applications that manage our existing suite of distribution channels is quite complex," says Fletcher. "It has grown over the course of 15 years. When the first GDS, OTA, and online travel agencies started popping up, we needed to allow people to book against our inventory systematically. The technological landscape was very different back then, and there's always been a lot of pressure to deliver on those systems, so they haven't necessarily gotten the care and feeding that they should have."

When the Google opportunity came along, however,

PEAK PERFORMANCE

Choice's Google Interface service handles up to 750,000 transactions a day and is capable of a peak rate of 540,000 transactions per hour.

More Than an App: AN ARCHITECTURE

The Google Interface project has given Choice Hotels International more than a new and innovative way to market its rooms to potential guests. It has also given the lodging company a new standard for its IT systems architecture: Java.

Before the Google Interface project, Choice's Phoenix, Arizona–based IT operations didn't have an overall systems architecture or even a systems architecture department, says Rain Fletcher, the company's vice president, application development and architecture. "There was never anybody who said, 'This is how we're going to develop software,' so developers were left to their own devices," Fletcher explains.

That approach may have worked in the past, but today Choice faces a complex, fast-changing business environment, fueled in large part by the Web and mobile devices. To stay on top, the company needs core IT systems that can adapt just as quickly.

To bring its systems up to date, Choice is migrating from its current legacy environment to a new service-oriented architecture (SOA), in which business processes are packaged as discrete reusable software components or enterprise services.

Because of the Google project, Choice's new SOA is based on Java, including the language and its related frameworks and APIs. In addition to the benefits of portability, scalability, and high performance, Java is especially suited to deliver on the promise of SOA environments. "Java has great support for SOA-based languages such as XPath, XML, and XSLT," notes Fletcher.

The proof is in the payoff. Choice is gaining four reusable enterprise services from the Google project: hotel distribution information, room availability, rate changes, and inventory. Two of these services—hotel distribution and room availability—are already being reused to power the company's push into mobile devices with its Choice Mobile application for Apple iPhone and Android–based devices.

Choice's IT staff quickly realized it had to think outside the box. "We didn't want to deliver the Google Interface on the existing platform," explains Fletcher. "The existing tools that we had used over the last 15 years to build those interfaces would have supported this also, although it would have been quite complex because of the unique requirements that Google presented. But we really didn't want to do that; we wanted to embrace our new architecture and start to abstract those other systems so that, hopefully, we can retire them one day soon."

Indeed, the set of Java tools, standards, and frameworks the Choice IT team put together to develop the Google Interface has become the company's IT architectural standard.

SPEEDY DELIVERY
Choice Hotels had less than three months to develop its Google Interface service.

THE STANDARD OF CHOICE

The Google team had three unusual technical requirements for Choice's new hotel information service. First, they wanted Choice to maintain the cache of data on Google's computers and systems, rather than feed it out to Google to cache. Second, Google wanted only a rolling 90-day supply of room and rate information—a mere subset of the data in Choice's CRS. And third, the information had to be updated every seven days or less, whether it had changed or not. "They assumed that the cache was stale after seven days, and they would stop exposing that rate and availability information," explains Fletcher.

Executives on the business side at Choice saw the project as an intriguing extension of Choice's distribution platform. "All of a sudden this was the most important project in the building," says Fletcher, adding that existing staff was requisitioned to work on the project in addition to their current workloads.

With all hands on deck and the support of Java and the Java platform, the Google Interface went live in mid-October last year. Today it is available to any Google Maps or Google Places user and can handle up to 750,000 transactions per day.

The Google Interface consists of three components, all written in Java. One component is a custom Java EE Connector Architecture (JCA) adapter that interfaces with Choice's CRS. The second component decides which information needs to get pushed out to Google, and the third, a scheduler, guarantees that the published data is never more than seven days old.

Scalability and performance were important considerations for choosing Java, but Choice had other reasons as well, says Fletcher. With Java, the company would achieve both hardware and software vendor independence. Choice already had Java expertise in-house—indeed, 80 percent of its IT staff already had some Java knowledge. Java talent is also widely available in the marketplace. Most important of all is Java's status as a premier language and development platform for enterprise applications.

"As a language, it is mature, well understood, and has a robust and expressive suite of powerful APIs," says Fletcher. The Google Interface project, in fact, uses a number of those APIs, including JCA and the Java Message Service (JMS).

"We made the decision to standardize on Java as our standard development platform specifically because of projects such as our Google Interface," Fletcher concludes, "which required complex logic, very high performance, and the ability to get to market quickly." </article>

Philip J. Gill is a San Diego, California–based freelance writer and editor.

Data Quality Tools for Java

Telephone Verification
Name Parsing & Genderizing
Geocoding
Address Verification & Standardization
Email Address Verification
Web Services & APIs
Duplicate Elimination
MELISSA DATA

BEFORE

john smith iii phd
melissa data corp.
22382 Empresa 92688
7145895200
john@800miAL.con

AFTER

Melissa Data Corp.
John Smith III PhD
22382 Avenida Empresa Ste 100
Rancho Santa Margarita, CA 92688-2112
949-589-5200
John@melissadata.com
Delivery Indicator: Business

*Highlights indicate added and/or corrected data.

Realtime NCOALink Change-of-Address Web Service available

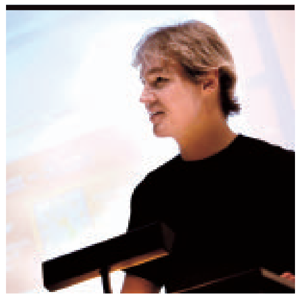
Now, finding the right data verification tools doesn't have to be so puzzling. Melissa Data offers customizable APIs, Web services and enterprise applications to match your budget and business needs. For solutions to cleanse, validate and standardize your contact data, we're ready to help you find the perfect fit.

Multiplatform

- Global address verification for 240 countries
- Clean and validate data at point-of-entry or in batch
- Correct misspellings, missing directionals, and confirm deliverability
- Enhance addresses with County, Census, FIPS, etc.
- Append rooftop lat/long coordinates to street addresses
- Update records with USPS and Canadian change of address info

Request free trials at MelissaData.com/myjava or call 1-800-MELISSA

MELISSA DATA®
Your Partner in Data Quality



MICHAEL KÖLLING

BIO

PHOTOGRAPH BY JOHN BLYTHE

Shall We Play a Game?

Improving and extending the code leads to the first playable version of our turtle game.

In the last two issues of *Java Magazine*, I introduced the beginning of a little computer game, written in Java using the Greenfoot environment. We created a keyboard-controlled turtle that walks around eating pizza.

In this article, we'll start making things more interesting by introducing a predator that's after our turtle—not only must our turtle collect pizza slices, but it must do so while avoiding being eaten by a snake!

You can download the project as it was at the end of our last article here. We will continue from where we left off, so if you missed the last articles, it might be a good idea to look at them first to see how we got this far.

Cleaning Up Our Code

The turtle's act method, as we left it last time, is shown in **Listing 1**. Before we go on to add more functionality to our program (and, in the process, make our code more complex), it is a good idea to take a minute to improve the structure of our code. Specifically, our act method is getting somewhat long, and it is starting to take some effort to read it and work out what it does.

In general, we should strive to keep our methods short and clear. We can do this by breaking the act method into multiple methods, each defining one logical task.

In our example, our act method does two logically distinct tasks:

- Movement (moving forward and possibly turning)
- Looking for pizza (and possibly eating it)

CLASS ACTION
You can find out about methods available in the **Greenfoot classes** by looking up the Greenfoot class documentation.

We can move each of these two tasks into a separate method, and then call each method from the act method. **Listing 2** illustrates this.

In this version of the code, we have defined two additional methods, moveAndTurn and eat. Have a good look at **Listing 2** to see how this was done.

Each of the method definitions has a section at the beginning containing some asterisks. For example, the eat method starts with this:

```
/**
 * Look for pizza and eat it
   if we see some.
 */
```

This is a *method comment*. A comment is written for the programmer (not for the computer). It explains, in plain English, the purpose of the method that follows.

The lines following the comment define the method. You will notice that the method bodies contain exactly the same code that was previously in the body of the act method in **Listing 1**.

Now, in **Listing 2**, the act method contains only calls to the two new methods:

```
public void act()
{
    moveAndTurn();
    eat();
}
```

Note that this version of our code does exactly the same thing as the previous version—it is functionally equivalent. However, because we have chopped it up into smaller pieces, it is now easier to read, and we can more easily recognize what the act method does.

You should make the same changes in your own code.

Snakes on a Plane!

Now that we have improved our code structure, we can move forward and add to our project. We now want to introduce an enemy of our turtle: snakes.

Start by creating another subclass of Actor, named Snake, with a snake image. (Remember, you can create an Actor subclass by right-clicking the Actor class and choosing **New subclass**. Name the class Snake and choose a snake image from the **Animals** category (see **Figure 1**.)

Compile your scenario and place a few snakes into your world (see **Figure 2**). You might want to save the world again (right-click the world background and choose **Save the World**) to add your snakes to the saved world.

Snake Behavior

Initially, our snakes don't do anything. We would like them to move around the screen randomly and try to eat the turtle. The goal for the player, then, is to control the turtle and eat all pizza slices while evading the snakes.

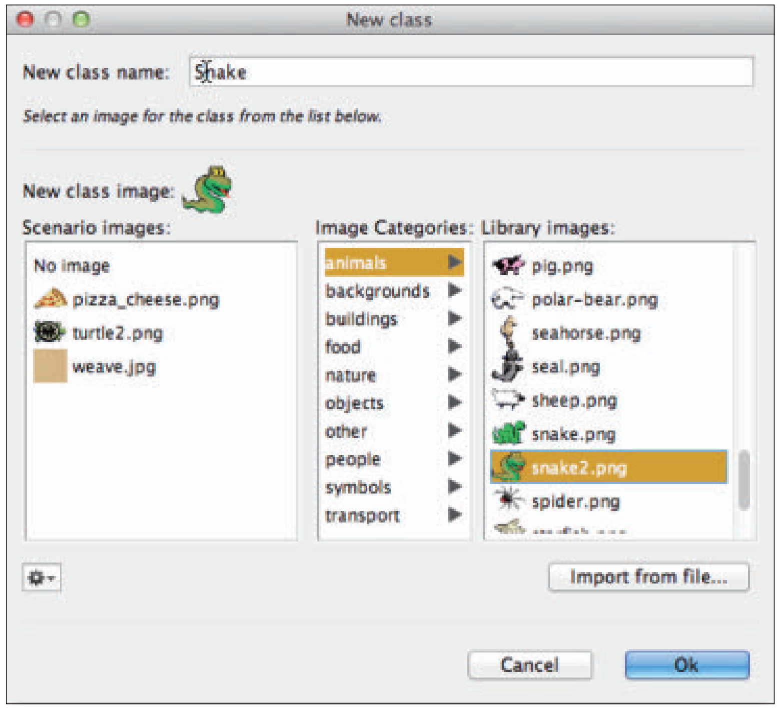
We can start by copying all three methods from the Turtle class (act, moveAndTurn, eat) into the Snake class. Do this by using copy and paste to replace the empty default act method in the Snake class. Make sure you watch the brackets—the class body still needs an opening and closing curly bracket.

At this point, you might like to compile and run your scenario to test what it does. It's not exactly what we want yet, but it does something. (You will see that the snakes follow your key presses, just as the turtle does.)

There are two things we'd like to change in the class Snake from the copied turtle behavior. First, snakes should eat turtles, not pizza. And second, snakes should move around on their own (randomly), rather than be controlled by the keyboard.

We can make the first change by going to the snake's eat method and replacing all mentions of "pizza" with "turtle," as shown in **Listing 3**.

Pay attention to capitalization: Uppercase and lowercase letters must be used exactly as shown. When writing *Turtle* (with a capital *T*), we are referring to the Turtle *class*. When writing in lowercase (*turtle*), we are referring to a *variable*.


New class

New class name: Snake

Select an image for the class from the list below.

New class image:

Scenario images: No image, pizza_cheese.png, turtle2.png, weave.jpg

Image Categories: animals, backgrounds, buildings, food, nature, objects, other, people, symbols, transport

Library images: pig.png, polar-bear.png, seahorse.png, seal.png, sheep.png, snake.png, snake2.png, spider.png

Import from file... | Cancel | Ok

Figure 1

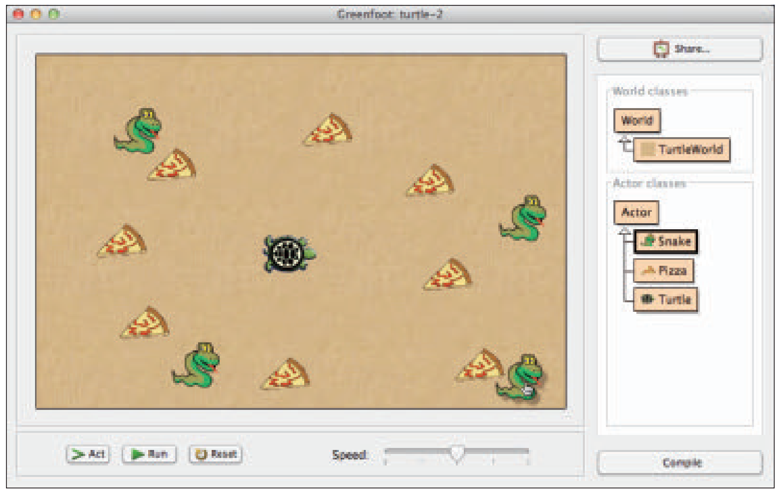


Figure 2

LISTING 1 | LISTING 2 | LISTING 3

```
public void act()
{
    move(4);

    if(Greenfoot.isKeyDown("left")) {
        turn(-3);
    }

    if(Greenfoot.isKeyDown("right")) {
        turn(3);
    }

    Actor pizza = getOneIntersectingObject(Pizza.class);
    if(pizza != null) {
        getWorld().removeObject(pizza);
    }
}
```

See all listings as text

Test your scenario. Snakes should eat turtles, not pizza.

Random Walking

At the moment, the snakes are still controlled by the same keyboard keys that steer the turtle. That's not good. We can change this, first by making snakes run in circles.

Remove the code from the body of the snake's moveAndTurn method that implements the keyboard-controlled turning, and replace it with a simple turn(3) statement. Your method should now look like this:

```
public void moveAndTurn()
{
  move(4);
  turn(3);
}
```

Try it out. Snakes are now moving on their own (even though it's in a very predictable pattern).

Now we want to make the movement more random. To do this, we can use a method from the Greenfoot class called getRandomNumber. This method will give us a random number between 0 and a specified limit. For example, the following will give us a random number between 0 and 19 (the specified limit itself is always excluded):

```
Greenfoot.getRandomNumber(20)
```

We can use this method to make the turns of the snake less predictable. If we use a random number as a parameter to the turn method (instead of the number *3* we used previously), the snake should walk more randomly:

```
turn(Greenfoot.
getRandomNumber(20));
```

If you try this, you will notice that the snake is still moving in a circle, albeit a somewhat irregular one. The reason is that the turns—even though somewhat random—are always right turns.

To fix this, we need to make the snake turn left as well. Turning left is done by turning at a negative angle. Instead of turning between 0 and 20 degrees, we would like to turn between minus 20 degrees and 20 degrees.

We can achieve this with the following change:

```
turn(Greenfoot.
getRandomNumber(40)-20);
```

That is, we first get a random number using 40 as the specified limit, and then we subtract 20 from it. The result will be between –20 and 20. (Actually, if you think carefully about it, this is slightly incorrect. The result will be between –20 and 19. Can you figure out why? Can you correct this? It is not actually very important in our context to correct this slight imprecision—the snake is moving erratically, so a little bias does not hurt much.)

Try this turn instruction (instead of the previous turn(3)) in your own code. You will see that the game suddenly becomes a lot more interesting!

Improving the Game

There are many ways in which you can improve this game. Instead of talking you through some of them right now, I'll show you something more useful: how you can find out more for yourself.

So far, I have introduced various useful methods that we could use, such as move and turn in the Actor class and getRandomNumber from the Greenfoot class. So how do you find out what other methods there are for you to use?

The answer lies in looking up the *class documentation*. Choose **Greenfoot Class Documentation** from Greenfoot's **Help** menu. This will open a Web browser that lists all the classes of the Greenfoot system. (You will see that there are six classes: Actor, World, Greenfoot, GreenfootImage, GreenfootSound, and MouseInfo.)

Click the Greenfoot class to see the documentation for this class. If you scroll down a bit on the page, you will see a list of methods available in this class. There are the getRandomNumber and isKeyDown methods that we have used already and a good number of other ones.

The format this is displayed in is called *javadoc*. It is a standard format for documenting Java methods, and this helps in determining what particular classes can do. The documentation includes the return type (what the method returns to you when you call it), the signature (that is, its name and parameters), and a comment explaining what the method does. Here, you can find out all about other available methods and how to call them.

For now, I'll leave you to explore these methods yourself. (The most interesting classes to look at initially are the Actor and Greenfoot classes.) In the next issue of *Java Magazine*, we will explore some more of them.

Tip: You can see and download many Greenfoot Java programs on the Greenfoot Website.

Conclusion

In this article, you learned a few specific things about Java code and some general principles:

- Code should be structured so that you have small methods and each does one logical task. (The act method itself should often contain only calls to other methods.)
- Random numbers can be used to create random behavior in your program (such as random movement).
- You can find out about methods available in the Greenfoot classes by looking up the Greenfoot class documentation.
- Class documentation is provided in javadoc format, which shows you information about all available classes and about each method within each class. </article>

LEARN MORE

- Java API
- "Getting Your Feet Wet" in the Premiere issue of *Java Magazine*

MAX BONBHEL

BIO

Part 3

Introduction to RESTful Web Services

RESTful Web services versus JAX-WS Web services: The face-off!

In Part 2 of this three-part series, I showed how to handle a RESTful Web service response to the client in JSON representation format. We also used HTML5, JavaScript, and Ajax to call the Web service, store the information locally, and display the information later.

Now, in Part 3, we will create a complete application based on a Java API for XML Web Services (JAX-WS) Web service and see in detail how to use a JavaServer Faces (JSF) client to access and consume the Web service. In the end, we will have two types of Web services in one application. This will allow us to compare and easily choose the type of Web service that best fits our needs.

Part 3 ends the series, but it is also a good segue to a new three-part series that will focus on the different aspects of Simple Object Access Protocol (SOAP) Web services, such as security, reliability, and transactions.

Note: The complete source code for the application designed in this article can be downloaded here.

QUICK WIZARD
We will use the **Web Service Client wizard** provided by the NetBeans IDE to create a Web service in five minutes.

What Are JAX-WS Web Services?

Unlike RESTful Web services, JAX-WS Web services are generally used for larger applications that demand several channels of communication among platforms, and they provide tools for integration and interoperability.

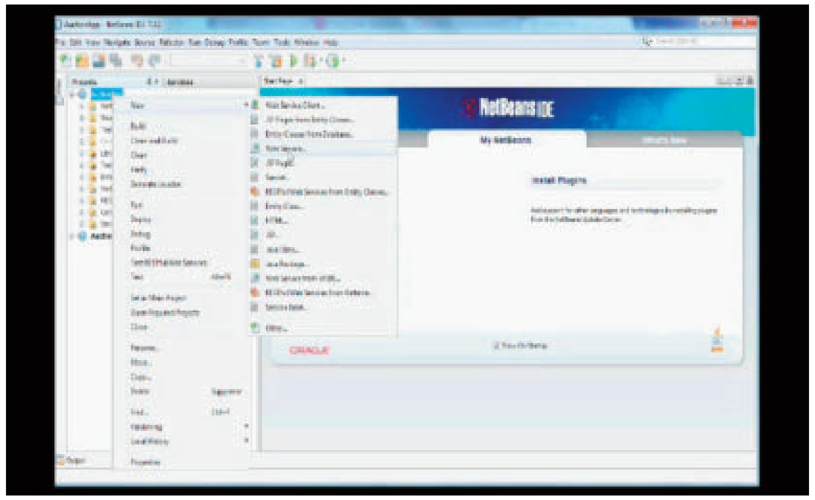
JAX-WS allows us to develop and deploy Web services easily using the Java platform. JAX-WS provides many protocols to facilitate communication among systems, such as SOAP (1.1 and 1.2), XML, HTTP, and more.

WSDL (Web Services Description Language) is an XML-based language for describing Web services and how to access them. A WSDL document is written in XML and specifies the location of the Web service and the operations (or methods) the service exposes.

SOAP, which is written in XML format, is one of the protocols used with Web services for enabling applications to exchange information over HTTP. In this article, we will see how SOAP uses XML for transmitting messages between applications.

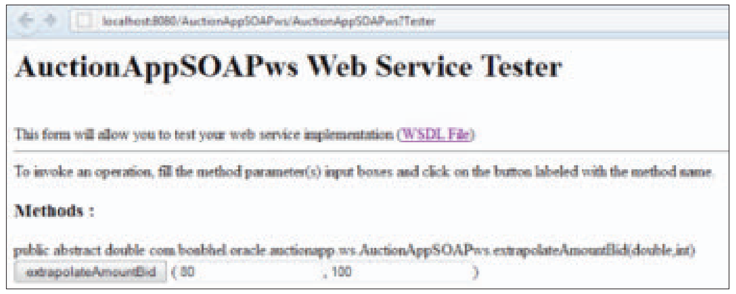
Prerequisites
The following software was used to develop the application described in this article:
- NetBeans, which can be downloaded here
- JAX-WS implementation (included in NetBeans), which is the Java EE specification (JSR-224) for building SOAP Web services
- AuctionApp, which can be downloaded here

Note: This article was tested using



In this screencast, Max Bonbhel provides an introduction to his article.

PHOTOGRAPH BY ALLEN MCINNIS/GETTY IMAGES


localhost:8080/AuctionAppSOAPws/AuctionAppSOAPws?Tester

AuctionAppSOAPws Web Service Tester

This form will allow you to test your web service implementation (WSDL File)

To invoke an operation, fill the method parameter(s) input boxes and click on the button labeled with the method name.

Methods :

public abstract double com.bonbhel.oracle.auctionapp.ws.AuctionAppSOAPws.extrapolateAmountBid(double,int)

extrapolateAmountBid (80 , 100)

Figure 1

NetBeans IDE 7.0.1, but as of this writing, the latest version of the NetBeans IDE is 7.1.

Real-Life Application for JAX-WS Web Services with NetBeans 7.0.1

In this practical section, we will build a real-life application step by step so that you can learn quickly the basics of the JAX-WS Web services.

We will use the NetBeans IDE for developing a typical enterprise application that will show you how to access and consume a JAX-WS Web service (an external service that resides in the application tier) from JSF client applications.

What we are going to do is add new capabilities to the AuctionApp we started creating in Part 1 of this series of articles by doing the following:

- Add a new JAX-WS Web service that extrapolates the amount of the bid.
- Generate the WSDL file.
- Create a Web service client to consume the JAX-WS Web service via a JSF application.

GO BIG

Unlike RESTful Web services, **JAX-WS Web services** are generally used for larger applications that demand several channels of communication among platforms, and they provide tools for integration and interoperability.

Let's Add and Test the JAX-WS Web Service

To begin, we need to add the SOAP Web service capability to our existing application.

1. Add the new Web service in the AuctionApp application:

   a. Open the AuctionApp in NetBeans IDE version 7 or later.

   b. Right-click the **AuctionApp** project node and choose **New**. Then select **Web Service**.

   c. Type AuctionAppSOAPws in the **Web Service Name** field and type com.bonbhel.oracle.auctionApp.ws in the **Package** field. Then click **Next**.

   d. Select the **Implement Web Service as a Stateless Session Bean** box in the upper part of the New Web Service dialog box.

   e. Click **Finish**.

   NetBeans generates the structure for the new Web service with a sample Web service and all the resources we need to operate our Web service.

   At this point, the source code for the Web service can be edited in the editor. Your AuctionAppSOAPws.java file should look like the code shown in **Listing 1**.

LISTING 1 LISTING 2

```
package com.bonbhel.oracle.auctionApp.ws;

import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.ejb.Stateless;

/**
 * @author Max Bonbhel
 */
@WebService(serviceName = "AuctionAppSOAPws")
@Stateless()
public class AuctionAppSOAPws {
  /** This is a sample web service operation */
  @WebMethod(operationName = "hello")
  public String hello(@WebParam(name = "name") String txt) {
    return "Hello " + txt + " !";
  }
}
```

See all listings as text

2. Now add a specific operation to the Web service that will extrapolate the amount of the bid by item:

   a. Open the **Web Services** node of the AuctionApp project and right-click the **AuctionAppSOAPws** node. Then select **Add Operation**.

   b. Type extrapolateAmountBid in the **Name** field and type double in the **Return Type** field.

   c. Click **Add** in the lower part of the Add Operation dialog box.

   d. Type amountBid in the **Name** field and type double in the **Type** drop-down list.

   e. Click **Add** again.

   f. Type factor in the **Name** field and type int in the **Type** drop-down list.

   g. Click **OK**.

3. Edit and modify the generated operation, as shown in **Listing 2**.

4. Test the JAX-WS Web services:

   a. Right-click the **AuctionApp** project and choose **Deploy**.

   b. Expand the **Web Service** node of the AuctionApp project. Then right-click the **AuctionAppSOAPws** node and choose **Test Web Service**.

   A tester page is displayed in your browser. If you see the page shown in **Figure 1**, it means your JAX-WS Web services are working correctly.

   c. Type two numbers in the tester page: 80 for amountBid and 100 for factor.

   d. Click **extrapolateAmountBid**.

AT YOUR SERVICE
JAX-WS allows us to develop and deploy Web services easily **using the Java platform.**

The result of the operation is displayed in **Figure 2**.

Let's Consume the JAX-WS Web Services with a JSF Client Application
It's time to create a Web client to consume the JAX-WS Web services we created. So we are going to quickly build a sample front-end JSF application to allow a user to perform the following tasks:

- Enter information for the seller, item, and bid.
- Extrapolate the amount of the bid using the JAX-WS Web service.
- Display the information.

We will use the Web Service Client wizard provided by the NetBeans IDE to generate the code and everything we need for looking up a Web service.

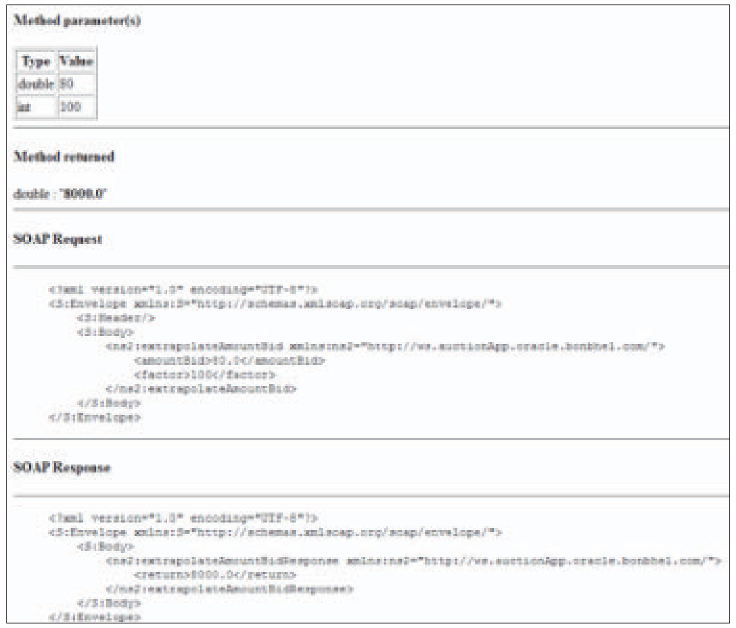
Let's code this application in five minutes.

1. Generate the initial NetBeans project:
   a. From the File menu, choose **New Project**.
   b. From Categories, select **Java Web**.
   c. From Projects, select **Web Application**.
   d. Click **Next**.
   e. Type a project name, AuctionAppWebServiceClient, and click **Next**.
   f. Make sure the server is GlassFish Server (or similar wording).
   g. Click **Finish**.
2. Create the entities based on the datasource we created in Part 1 of this series:
   a. Right-click the **AuctionAppWebServiceClient** project and select **New**. Then select **Entity class from Database**.
   b. In the Databases Tables dialog box, select your datasource in the **Data Source** drop-down list.
   c. In the upper part of the Databases Tables dialog box, select the **BID**, **SELLER**, and **ITEM** tables and click **Next**.
   d. In the Entity Classes dialog box, type a **Package name**, such as com.bonbhel.oracle.auctionAppWebServiceClient.
   e. Accept all other default settings, and then click **Finish**.

   NetBeans generates the Seller.java, Item.java, and Bid.java files.
3. The client implementation we are going to build consists of JSF pages based on the entities just created, so create the JSF pages:
   a. Right-click the **AuctionAppWebServiceClient** project and select **New**. Then select **JSF Pages from Entity Classes**, click **Add all**, and click **Next**.
   b. Type a **Session Bean Package** name, such as com.bonbhel.oracle.auctionAppWebServiceClient.facade, and type a **JSF Classes Package** name, such as com.bonbhel.oracle.auctionAppWebServiceClient.controller.
   c. Type a **Folder Name**, such as jsfClient, and click **Finish**.
   d. Click **Finish**.
4. Use the Web Service Client wizard to create the Web service client. (We will assume that the JAX-WS Web service is an external service that resides in the application tier over the network. So, we will use the URL to the JAX-WS Web service WSDL file.)
   a. Make sure the AuctionApp we started creating in Part 1 of this series is up and running. If it is not, right-click the **AuctionApp** node and choose **Deploy**.
   b. Open the **Web Service** node of the AuctionApp project and right-click the **AuctionAppSOAPws** node. Then select **Properties**.
   c. Notice that the URL of the WSDL file is shown in the upper part of the Properties dialog box. You need it for a later step.
   d. Right-click the **AuctionAppWebService-Client** node and choose **New**. Then select **Web Service Client**.
   e. In the New Web Service Client wizard, specify the URL to the Web service WSDL file: http://localhost:8080/AuctionAppSOAPws/AuctionAppSOAPws?wsdl.
   f. Accept all other default settings. The package name will be taken from the WSDL file.
   g. Click **Finish**.

   NetBeans generates the structure of the new Web service client, as shown in **Figure 3**.

Note: If the WSDL file cannot be downloaded, click **Set Proxy** and then specify the proxy server.



Method parameter(s)

Type	Value
double	80
int	100

Method returned

double : "8000.0"

SOAP Request

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Header/>
    <S:Body>
        <ns2:extrapolateAmountBid xmlns:ns2="http://ws.auctionApp.oracle.bonbhel.com/">
            <amountBid>80.0</amountBid>
            <factor>100</factor>
        </ns2:extrapolateAmountBid>
    </S:Body>
</S:Envelope>
```

SOAP Response

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Body>
        <ns2:extrapolateAmountBidResponse xmlns:ns2="http://ws.auctionApp.oracle.bonbhel.com/">
            <return>8000.0</return>
        </ns2:extrapolateAmountBidResponse>
    </S:Body>
</S:Envelope>
```

Figure 2

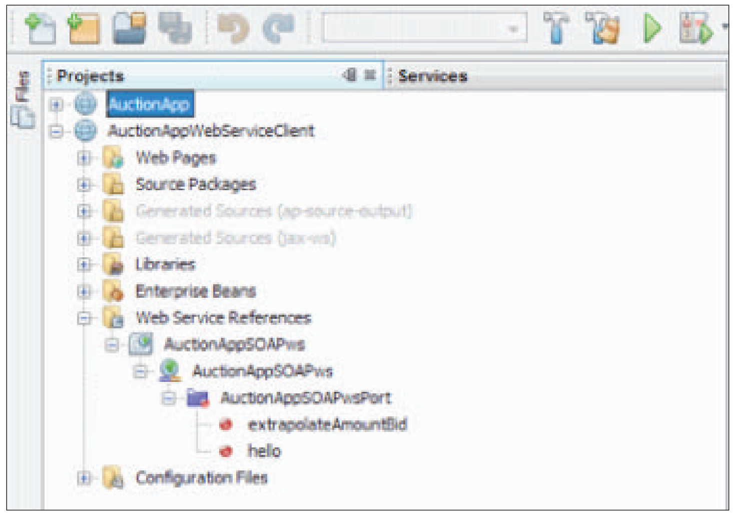
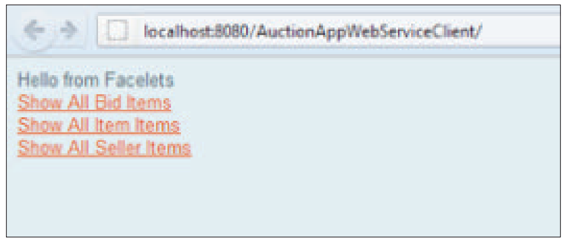


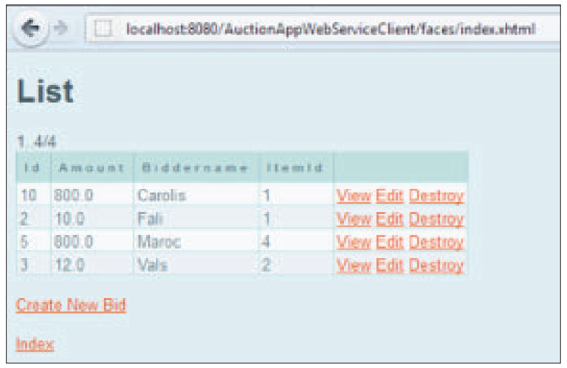
Figure 3

5. Add the extrapolateAmountBid operation provided by the JAX-WS Web service to the client classes:
   a. Open the **Source Packages** node of the AuctionAppWebServiceClient project and double-click the BidController.java file located in the controller package com.bonbhel.oracle.auctionAppWebServiceClient.controller.


localhost:8080/AuctionAppWebServiceClient/

Hello from Facelets
Show All Bid Items
Show All Item Items
Show All Seller Items

Figure 4


localhost:8080/AuctionAppWebServiceClient/faces/index.xhtml

List

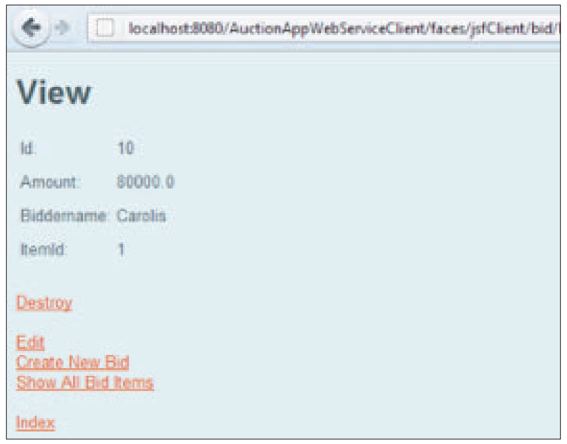
1..4/4

Id	Amount	Biddername	Itemid	
10	800.0	Carolis	1	View Edit Destroy
2	10.0	Fali	1	View Edit Destroy
5	800.0	Maroc	4	View Edit Destroy
3	12.0	Vals	2	View Edit Destroy

Create New Bid

Index

Figure 5


localhost:8080/AuctionAppWebServiceClient/faces/jsfClient/bid/

View

Id:	10
Amount:	80000.0
Biddername:	Carolis
Itemid:	1

Destroy

Edit
Create New Bid
Show All Bid Items

Index

Figure 6

   b. Put your cursor anywhere inside the source editor.
   c. Expand the **Web Service References** node of the AuctionAppWebServiceClient project and drag the **extrapolateAmountBid** node inside the source editor.

Note: Alternatively, you can right-click anywhere in the source editor and choose **Insert Code**. Then select **Call Web Service Operation** and click the **extrapolateAmountBid** operation in the Select Operation to Invoke dialog box.

6. The extrapolateAmountBid() method appears at the end of the BidController class code, as shown in **Listing 3**.
7. We need to add some application logic in the BidController class in order to extrapolate the amount of the bid (by factor, which is 100) when the user is editing or viewing the bid entry. So call the extrapolateAmountBid operation to extrapolate the bid:
   a. Open the BidController.java file in the source editor.
   b. Modify the public String prepareView() method, as shown in **Listing 4**.

Testing the Client Application

1. Run the JSF client application:
   a. Make sure the AuctionApp we started creating in Part 1 of this series is up and running. If it is not, right-click the **AuctionApp** node and choose **Deploy**.
   b. Right-click the **AuctionAppWebServiceClient** project and choose **Run**.

   The list of all entries is displayed, as shown in **Figure 4**.
2. Invoke the JAX-WS Web service to display the extrapolated amount of the bid:
   a. Click the **Show all Bid Items** link to display the list of bid entries, as shown in **Figure 5**.
   b. Click the **View** link of any Bid item to see the newly extrapolated amount of the bid. In our case it's Carolis' Bid, as shown in **Figure 6**.

      As you can see, the amount of the bid changed from 800.0 to 80000.0.
   c. Now you can smile, because it works!

LISTING 3 LISTING 4

```
private double extrapolateAmountBid
(double amountBid, int factor) {
com.bonbhel.oracle.auctionapp.ws.AuctionAppSOAPws
port = service.getAuctionAppSOAPwsPort();
return port.extrapolateAmountBid(amountBid, factor);
}
```

See all listings as text

Conclusion

We have seen how easy it is to configure a JAX-WS Web service and consume it from a JSF application client. The integration of RESTful Web services and JAX-WS Web services in the same application allowed us to compare them so we can choose the type of Web service that best fits our needs.

NetBeans provide many tools and services for creating Web services for the client side and for consuming and accessing over the network external services that reside in the application tier. </article>

LEARN MORE

- NetBeans documentation, training, and support
- "Developing JAX-WS Web Service Clients"
- "Binding WSDL to Java with JAXB"
- "Advanced Web Service Interoperability"
- "Web Services Learning Trail"
- Oracle University: Developing Web Services Using Java Technology, Java EE 6

Cloud/Java EE

Tame the Cloud

Oracle's **Cameron Purdy** on how Java EE is evolving to become the cloud computing platform standard. **BY MICHAEL MELOAN**

The enterprise landscape is increasingly complex, with information flowing from more sources than ever before. The emphasis is shifting from standalone applications to dynamic shared environments that will enhance scalability, automation, and efficiency. Cloud computing promises to be a transformational technology for Java EE. Cameron Purdy, vice president of Java EE development at Oracle, provides an overview of the key issues in this arena and how they will affect enterprise developers and customers.

PHOTOGRAPHY BY DAVE BRADLEY
ART BY I-HUA CHEN

Java Magazine: How will cloud computing deliver advantages in the enterprise space?
Purdy: At this fairly early stage in the development of cloud technology, the major focus is on PaaS [platform-as-a-service] capability. A number of infrastructure cloud vendors offer dozens or even hundreds of servers as easily as buying something on an e-commerce site. This is a move toward decentralization and democratization of IT infrastructure. It offers abstraction of hardware and software elements all the way up to development environments and middleware components, which

frees developers to focus on enterprise solutions. This approach offers real benefits in terms of operational simplicity, time to market, and cost savings.

At Oracle, we'll be looking at how PaaS will affect different roles within an organization, such as development, QA, operational staging, and actual production operations. Then we need to integrate these considerations into our own suite of products, as well as work with our industry partners to standardize the model, the APIs, and the overall Java EE platform specifications.

Java Magazine: Can you provide an example of how Oracle is reflecting these considerations in its products?

Purdy: Sure. For example, within Oracle WebLogic Server 12*c*, we provide support for dependency injection, the Java Persistence API [JPA], and a uniform build process via the Maven WebLogic Plug-in, as well as for Agile development methodology in concert with Hudson Continuous Integration. These standards make enterprise applications easier to build as well as operate. Significant improvements have also been realized in managing and achieving high performance in virtualized environments, which is a prerequisite in PaaS environments. At Oracle, we deliver these optimizations with Oracle VM.

Java Magazine: What are the most important aspects to consider when implementing a cloud-oriented enterprise architecture?

Purdy: There are three key concepts related to deploying enterprise PaaS solutions: availability, elasticity, and multitenancy. In general, applications will be running on virtual machines in a remote environment controlled through a Web-based console. Because there is no physical access to a local system, the ability to mitigate problems is dramatically reduced. To ensure availability, systems need to be more autonomic. They need to be self-healing. One of the ways to achieve this resiliency is through redundancy.

PRIMED FOR CLOUD

The container-based architecture and its inherent abstraction of resource access make the Java EE platform **well suited for cloud computing.**

(From left to right) Oracle's Mac Hale, director of software development; Cameron Purdy, vice president of Java EE development; and Jim Gish, consulting member of technical staff, talk shop.

Oracle's Mike Lehmann and Arun Gupta discuss Oracle WebLogic Server 12c at Oracle OpenWorld Latin America 2011 in São Paolo, Brazil.

Maintaining availability, even in the event of hardware failure or electrical failure, requires that the application be running in multiple places, and possibly in multiple data centers, to prevent a single point of failure.

Availability is also facilitated through the concept of elasticity. Applications need to be robust enough to seamlessly operate while a given server is down for maintenance or due to failure. They must also be able to respond to additional capacity demands from increases in concurrent users or fluctuations in processing load. Elastic applications must be able to add resources dynamically, which fits very well with the cloud computing metaphor.

The concepts of availability and elasticity are both in concert with enterprise architectures built around PaaS and IaaS [infrastructure as a service].

Multitenancy is also an important principle in cloud computing. It refers to the ability to collocate multiple users within a single environment. A CRM [customer relationship management] system, for example, would need to provide service to a number of different companies. Each company represents a different tenant. And those tenants could be supported by dedicated hardware, or virtualized, providing each tenant with a virtual machine or a set of virtual machines within a given hardware infrastructure. There are many levels of multitenancy, even down to multiple tenants inside a single JVM [Java Virtual Machine].

There are trade-offs across the board in multitenant architectures. Security, resource utilization, and cost issues are at the forefront. Creating isolation and protection within a JVM is an important area of investment right now. We are also looking at how to meter CPU and memory demands by a particular tenant. Managing and restricting how many resources a tenant can consume is one of the fundamental challenges of multitenant systems. We need to make sure that a tenant can't chew up an entire CPU or use so much memory that a system is brought to its knees. We want to provide building block technologies to manage these issues for our customers.

As we revisit each aspect of Java EE, multitenancy, availability, and elasticity will all be important considerations in planning with our partners for the future of the platform.

Java Magazine: What new capabilities will the Java EE 7 release offer?

Purdy: We are working very closely with our industry partners to define and deliver Java EE 7. The major theme in Java EE 6 was ease of use, and that continues in Java EE 7. But the prime mover for Java EE 7 will be the cloud. The container-based architecture and its inherent abstraction of resource access make the Java EE platform well suited for cloud computing. This approach ensures that portable applications can target single-machine deployments as well as large cluster installations without fundamental changes to the programming model. The move to cloud computing is a further evolution of this powerful paradigm.

Multitenancy will be supported via virtual servers mapping requests to a tenant. And there will be more emphasis on CDI [Contexts and Dependency Injection], the JPA,

In Oracle's Burlington, Massachusetts, office, Oracle Architect Gene Gleyzer updates Purdy on his current projects.

Dave Carrano, software engineer, shares his work with Purdy; Alex Gleyzer, vice president of development, talks with Purdy about Java EE 7.

JAX-RS [Java API for RESTful Web Services], JSF, Servlets, and EJBs.

We also expect the Java EE 7 platform to add support for new and evolving technologies in the Web space, including WebSockets and HTML5, and a modern HTTP client API as defined by JAX-RS 2.0.

Modularity and versioning capabilities are also being investigated. This work will be coordinated with the upcoming modularity JSR for Java SE, to ensure alignment across the Java SE and Java EE platforms.

Java Magazine: Modularity and lambda expressions are major themes for Java SE 8. How will Java EE benefit from these developments?

Purdy: We see a broad set of possibilities for these capabilities. Modularity is an important area of focus for Oracle toward Java EE. It will yield benefits in scalability, robustness, and maintainability. Project Jigsaw is the OpenJDK effort to design and implement a standard module system for the Java SE platform. Its implementation in Java SE 8 will provide an instrumental building block for evolving the Java enterprise platform.

We've already been able to leverage the work of the OSGi [Open Services Gateway initiative] to build modularity into Oracle GlassFish Server and GlassFish Server Open Source Edition. With Project Jigsaw coming to Java SE, we will continue to proliferate this approach with standards like OSGi and integrate these efforts very closely with capabilities supported by the JVM.

Regarding lambda expressions, they're typically useful in implementing aggregate operations that can frequently be parallelized using multicore processors. We've seen opportunities for leveraging this in technologies like JDBC, JPA, EJBs, JMS filtering, and many other areas. Lambda expressions have been powerful in other languages and environments, and we're looking forward to supporting this approach in the Java EE platform.

Java Magazine: Any closing observations on the importance of the cloud for the future of enterprise systems?

Purdy: I believe we're on the cusp of a new wave of innovation. We saw this roughly 15 years ago, with a surge of development around Web-oriented technologies. And 10 years ago, with the release of J2EE, we saw the rise of business applications moving to the Java EE platform. Right now it's happening again, as incredible investments are being made to take advantage of what cloud technology can provide in terms of cost savings and flexibility.

Concurrently, we're moving toward significantly enhanced user experiences that will be provided by HTML5. This confluence of important new technologies represents a unique opportunity to evolve the Java Enterprise platform and bring new partners, organizations, and developers into the arena. It will powerfully expand the technological landscape and the entire enterprise market.

I expect to see wonderful levels of innovation as we begin to explore the possibilities of HTML5 and cloud technology. It's incredibly exciting to be a part of it. </article>

Michael Meloan began his professional career writing IBM mainframe and DEC PDP-11 assembly languages. He went on to code in PL/I, APL, C, and Java. In addition, his fiction has appeared in *WIRED*, *BUZZ*, *Chic*, *LA Weekly*, and on National Public Radio.

LEARN MORE

- Java EE at a Glance
- Oracle WebLogic Server

ARUN GUPTA

BIO

PHOTOGRAPH BY MARGOT HARTFORD

Cloud/Java EE

Looking Ahead to Java EE 7

Get educated about Java EE 7 JSRs.

With the release of Java EE 7 scheduled for the second half of 2012, projected JSRs are all up and running. The Java EE 7 release, which will reflect the evolving needs of the industry as it moves into the cloud, is date driven: anything not ready will be deferred to Java EE 8.

Here is an update and summary of the key features of different specifications in the Java EE 7 platform.

Java EE 7 Specification (JSR-342)
- Main theme: to easily run applications on private or public clouds
- The platform will define an application metadata descriptor to describe the PaaS execution environment such as multi-tenancy, resources sharing, quality of service, and dependencies between applications
- Embrace latest standards like HTML5, WebSocket, and JSON and have a standards-based API for each one of them
- Remove inconsistencies between Managed Beans, EJBs, Servlets, JSF, CDI, and JAX-RS
- Possible inclusion of JAX-RS 2.0 in the Web Profile, revised JMS 2.0 API
- Technology refresh for several existing technologies and possible inclusion of Concurrency Utilities for Java EE (JSR-236) and JCache (JSR-107)
- Status:
  - Approved by the JCP
  - Spec leads: Linda DeMichiel, Bill Shannon (Oracle)
  - Project page
  - Mailing list archive, jsr342-expert@javaee-spec.java.net, users@javaee-spec.java.net

Java Persistence 2.1 (JSR-338)
- Support for multitenancy
- Support for stored procedures and vendor function
- Update and Delete Criteria queries
- Support for schema generation
- Persistence Context synchronization
- CDI injection into listeners
- Status:
  - Approved by the JCP
  - Spec lead: Linda DeMichiel (Oracle)
  - Project page
  - Mailing list archive, jsr338-experts@jpa-spec.java.net, users@jpa-sepc.java.net

JAX-RS 2.0: The Java API for RESTful Web Services (JSR-339)
- Client API—low level using builder pattern and possibly a higher level on top of that
- Hypermedia—easily create and process links associated with resources
- Form or Query parameter validation using Bean validation
- Closer integration with @Inject
- Server-side asynchronous request processing
- Server-side content negotiation using "qs"
- Status:
  - Approved by the JCP, Early Draft available, Draft Javadocs
  - Spec leads: Santiago Pericas-Geersten, Marek Potociar (Oracle)
  - Project page
  - Mailing list archive, jsr339-experts@jax-rs-spec.java.net, users@jax-rs-spec.java.net

Java Servlet 3.1 Specification (JSR-340)
- Optimize the PaaS model for Web applications
- Multitenancy for security, session, and resources
- Asynchronous IO based on NIO2
- Simplified asynchronous Servlets
- Utilize Java EE concurrency utilities
- Enable support for WebSockets
- Status:
  - Approved by the JCP
  - Spec leads: Shing-Wai Chan, Rajiv Mordani (Oracle)
  - Project page
  - Mailing list archive, jsr340-experts@servlet-spec.java.net, users@servlet-spec.java.net

Expression Language 3.0 (JSR-341)
- Separate ELContext into parsing and evaluation contexts
- Customizable EL coercion rules
- Reference static methods and members directly in EL expressions
- Adding operators like equality, string concatenation, and size of

- Integration with CDI such as generating events before/during/after the expressions are evaluated
- Status:
  - Approved by the JCP
  - Spec lead: Kin-man Chung (Oracle)
  - Project page
  - Mailing list archive, jsr-341-experts@el-spec.java.net, users@el-spec.java.net

Java Message Service 2.0 (JSR-343)

- Ease of development—changes to the JMS programming model to make the application development simpler and easier
- Remove/clarify ambiguities in the existing specification
- Integration with CDI
- Clarification of the relationship between JMS and other Java EE specs
- A new mandatory API to allow any JMS provider to be integrated with any Java EE container
- Multitenancy and other cloud-related features from the platform
- Status:
  - Approved by the JCP
  - Spec lead: Nigel Deakin (Oracle)
  - Project page
  - Mailing list archive, jsr-343-experts@jms-spec.java.net, users@jms-spec.java.net

JavaServer Faces 2.2 (JSR-344)

- Ease of development—making configuration options dynamic, make cc:interface in composite components optional, shorthand URLs for Facelet tag libraries, integration with CDI, OSGi support for JSF artifacts
- Support implementation of Portlet 2.0 Bridge (JSR-329)
- Support for HTML5 features like HTML5 Forms, Metadata, Heading and Section content model
- Flow management, listener for page navigation events, and new components like FileUpload and BackButton
- Status:
  - Approved by the JCP, Early Draft available
  - Spec lead: Ed Burns (Oracle)
  - Project page
  - Mailing list archive, jsr344-experts@javaserverfaces-spec-public.java.net, users@javaserverfaces-spec-public.java.net

Enterprise JavaBeans 3.2 (JSR-345)

- Enhancements to the EJB architecture to enable PaaS, such as multitenancy
- Factorization of container-managed transactions to use outside EJB
- Further use of annotations
- Alignment and integration with other specifications in the platform
- Status:
  - Approved by the JCP
  - Spec lead: Marina Vatkina (Oracle)
  - Project page
  - Mailing list archive, jsr-345-experts@ejb-spec.java.net, users@ejb-spec.java.net

Contexts and Dependency Injection for Java EE 1.1 (JSR-346)

- Global ordering of interceptors and decorators
- API for managing built-in contexts
- Embedded mode to allow startup outside Java EE container
- Declarative control over which packages/beans are scanned in an archive
- Injection for static members such as loggers
- Send Servlet events as CDI events
- Status:
  - Approved by the JCP, Early Draft available
  - Spec lead: Pete Muir (RedHat)
  - Project page
  - Mailing list archive, twitter feed

Bean Validation 1.1 (JSR-349)

- Integration with other Java EE specs
  - JAX-RS: Validate parameters and return values on HTTP calls
  - JAXB: Convert constraints into XML schema descriptor
- Method level validation
- Apply constraints on group collection
- Extend the model to support AND and OR style composition
- Status:
  - Approved by the JCP
  - Spec lead: Emmanuel Bernard (RedHat)
  - Project page
  - Mailing list archive

JCACHE: Java Temporary Caching API (JSR-107)

- API and semantics for temporary, in-memory caching of Java objects, including object creation, shared access, spooling, invalidation, and consistency across JVMs
- Package: javax.cache
- Status:
  - Approved by the JCP
  - Spec leads: Yannis Cosmadopoulos (Oracle), Cameron Purdy (Oracle), and Gregory Luck (Software AG)
  - Project page
  - Mailing list archive

Java State Management (JSR-350)

- API that can be used by applications and Java EE containers to offload the responsibility of statement management to third-party providers with different QoS characteristics
- Java SE–based callers can access the state data by querying the state providers
- Providers with different QoS can be added, and API callers can query to meet their criteria
- Package: javax.state and javax.state.provider
- Status:
  - Approved by the JCP
  - Spec lead: Mitch Upton (Oracle)
  - Project page
  - Mailing list archive, jsr-350-experts@java-state-management.java.net

Batch Applications for the Java Platform (JSR-352)

- Programming model for batch applications and a runtime for scheduling and executing jobs
- Defines Batch Job, Batch Job Step, Batch Application, Batch Executor, and Batch Job Manager for the standard programming model
- Package: javax.batch

- Status:
  - Approved by the JCP
  - Spec lead: Chris Vignola (IBM)
  - Project page
  - Mailing list archive, issues@jbatch.java.net

Concurrency Utilities for Java EE (JSR-236)
This JSR has been ongoing for several years and should be revived and completed in time for Java EE 7.

- Provides a clean, simple, independent API by building on JSR-166, making it appropriate for use within any Java EE container
- Package: javax.util.concurrent
- Status:
  - Approved by the JCP
  - Spec leads: Anthony Lai, Naresh Revanuru (Oracle)
  - Project page: TBD
  - Mailing list archive: TBD

Java API for JSON Processing (JSR-353)
The Java API for JSON Processing JSR has now been filed as JSR-353.

- Status:
  - Approved by the JCP
  - Spec lead: Jitendra Kotamraju (Oracle)
  - Project page
  - Mailing list archive

The expert groups (EGs) for most of the JSRs have already been formed, but you can still participate by joining the publicly visible aliases and reviewing the drafts. The Oracle-led JSRs are all running transparently with dedicated projects on Java .net enabling, for instance, public access to EG discussions and thus anticipating the requirements set by the newly voted JCP 2.8 rules.

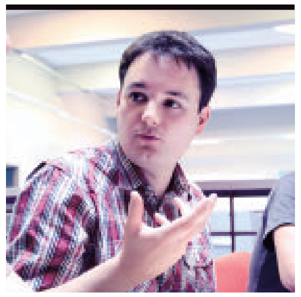
All the JSRs following JCP 2.8 are run more transparently. Here are some JCP 2.8 highlights:

- Names of the EG members are publicly visible.
- EG business is reported on a publicly readable alias.
- The schedule is public, current, and updated regularly.
- The public can read/write to a wiki to discuss the status so far.
- There is a discussion board on jcp.org.
- There is a public read-only issue tracker.

All of this will be integrated in GlassFish—the reference implementation for Java EE 7.
</article>

LEARN MORE

- Java EE Platform Specification
- GlassFish



JULIEN PONGE

BIO

Part 1

Project Coin: The Java Language Has Evolved!

Three additions to the Java language in Java SE 7 will help you be more productive.

Project Coin is an OpenJDK project that initiated the Java language changes that were standardized as part of Java SE 7 under JSR-334 in the Java Community Process (JCP). The project is self-described as follows:

"The goal of Project Coin is to determine what set of small language changes should be added to JDK 7. That list is:

- *Strings in switch*
- *Binary integral literals and underscores in numeric literals*
- *Multi-catch and more precise rethrow*
- *Improved type inference for generic instance creation (diamond)*
- *try-with-resources statement*
- *Simplified varargs method invocation"*

This article is the first in a two-part series covering each item of Project Coin. The goal is to present the immediate usage of each feature, and also to provide some background on the implications for implementing each of them, especially since they all maintain backward compatibility with prior versions of Java SE. The impact of each change was remarkably balanced with a scientifically sound analysis of millions of lines of existing Java code.

STRING SWITCH
Java SE 7 allows **strings** to be used in **switch statements.**

Part 1 focuses on the addition of binary integral literals, underscores in numeric literals, strings in switch statements, and type inference for generic instance creation.

Note: The source code for the examples described in this article can be downloaded here.

Binary Integral Literals and Underscores

Prior to Java SE 7, numeric literals could be specified either in *decimal*, *octal*, or *hexadecimal* forms, such as in the following:

```
/* The value 123 in decimal,
octal and hexadecimal */
byte decimal = 123;
byte octal = 0173;
byte hexadecimal = 0x7b;
```

The rules are simple. A decimal number is a literal comprising a combination of digits between 0 and 9, an octal number starts with a zero, and a hexadecimal number starts with 0x. With the advent of Java SE 7, you can now specify a number as a binary literal by starting it with 0b followed by a combination of zeros and ones up to the size of the underlying primitive type. The decimal number 123 can be written as the following binary literal:

```
// 123 in binary
byte binary = 0b01111011;
```

If fewer digits are provided than expected for the underlying primitive type, the missing digits are considered to be at the beginning of the representation, and they are given a value of zero. Thus, the

Julien Ponge provides a brief introduction to his article about Project Coin.

PHOTOGRAPH BY
MATT BOSTOCK/GETTY IMAGES

previous example could be written with one less digit:

```
// 123 in binary
byte binary = 0b1111011;
```

There are arguably a few readability problems in literals as the number of digits increases. While hexadecimal and binary literals start with a clear prefix (0x and 0b), octal literals start with a zero, which can be confusing. Indeed, it is naturally tempting to interpret 0173 as 173 instead of 123. In a bid to increase readability, Java SE 7 adds the ability to introduce underscores as part of numeric literals. As an example, 1200645790 can be written as 1_200_645_790. Back to the examples based on the number 123, we can rewrite them as follows:

```
byte decimal = 123;
byte octal = 0_173;
byte hexadecimal = 0x7B;
byte binary = 0b0111_1011;
```

Of course, underscores can also be used in float, double, and long numeric literals. A double number ends with D, while a long integer ends with L. Thus, the following literals can benefit from underscores:

```
double doubleValue = 1.111_222_444D;
long longValue = 1_234_567_890L;
long longHexa = 0x1234_3b3b_0123_cdefL;
```

There are, however, certain rules to respect in the usage of underscores. While they can be repeated at will (1______2___3 is the same as 123 or 1_2_3), they can appear only between digits. They cannot be put at the beginning or at the end of a numeric literal, before or after the point in a floating-point number, in or after a binary or hexadecimal literal prefix, and before or after a long (L) or double (D) suffix. Hence, the following literals raise a compiler error:

```
int a = _123_;
long b = 123_L;
int c = 0x_abc;
int d = 0_x_abc;
double e = 1000_._666_666_D;
```

Possible correct literals would be

```
int a = 123;
long b = 1_2_3L;
int c = 0xa_b_c;
int d = 0xa_bc;
double e = 1000.666_666D;
```

This change in the specifications of the Java language is transparent to the Java Virtual Machine and existing bytecode and libraries, because the Java compiler simply ignores underscores. Similarly, the new binary literals are treated as a new case beside octal and hexadecimal literals.

In summary, these changes are lightweight yet beneficial for improving the readability of code manipulating numeric literals.

Strings in Switch Statements

In the Java programming language, the switch control-flow statement works with integer values, such as in **Listing 1**.

LISTING 1 LISTING 2 LISTING 3

```
private static void basicSwitch(int value) {
  switch (value) {
    case 2:
      System.out.println("Number 2!");
    case 1:
    case 3:
      System.out.println("A number in the [1, 3] range: " + value);
      break;
    default:
      System.out.println("A number outside the [1, 3] range: " + value);
  }
}
```

See all listings as text

In **Listing 1**, a message is printed for cases where value is equal to 1, 2, or 3. A special case exists when value is 2, because a message is printed before passing over the execution to the handling cases of values 1 and 3. Other cases are collected as part of the default case. The switch statement is not limited to int. It also works for char, byte, and short, as well as the object wrapper types Character, Byte, Short, and Integer. Java SE 5 introduced the support of enumerated types, and they can also be used in switch statements. **Listing 2** illustrates switches on enumerated types.

Switching on enumerated types is based on the value returned by the ordinal() method of java.lang.Enum, the base class of all enumerated types, and that returns an integer. Hence, such a switch statement is still based on an integer value that is arbitrarily assigned by the Java compiler to each enumerated value.

Java SE 7 allows strings to be used in switch statements, too. This is interesting, because strings are often used in conditions for control-flow branching. **Listing 3** is an example reflecting the use of strings in switch statements.

A NullPointerException is thrown if the value to be switched on is the null value. Also, the case statements need to be on string literals or any constant expression of java.lang.String type.

The addition of strings in switch statements did not require any modification to the Java Virtual Machine bytecode. Rather, this functionality was implemented as syntactic sugar, where the compiler infers semantically equivalent code, but it is based on integer switches.

In order to understand how this works, we can use the javap decompiler tool, which ships as part of the Java SE Development Kit, and analyze the generated bytecode for the isTrue() method above. The equivalent Java code is shown in **Listing 4**.

Java compilers are free to implement switch statements with strings differently as long as the semantics are preserved. The string switched on is compared for equality against the case label. The generated code starts by a first switch statement based on the hashCode() value of the string constants being used. This acts as a way to directly jump to the comparison between the switched string and the possible case constants.

Note that the hashCode() value alone is not enough to compare strings, because the risk of collisions between different strings having the same hashCode() value is too high. It is even relatively easy to construct a string with a given hashCode() value (see Joe Darcy's post, Unhashing a String). When a string is matched, an integer is set based on the index of the original case statement, and it is used in the subsequent switch statement that contains the original instructions intended for the string case statements.

Improved Type Inference for Generics

Generics were added in Java SE 5 to increase the type safety in programs. They are especially useful when dealing with Java collections. **Listing 5** shows a typical usage of generics on collections.

There is, however, a strong sense of ceremony in this code. Let us just consider the declaration of strings and replace the code with the following:

```
List<String> strings =
new LinkedList<Integer>();
```

Of course, this code does not compile. There is a necessary concordance between the parametric types in a variable declaration and instance creation. Having to repeat explicit types on each side of such assignments is considered to be redundant by most developers.

Java SE 7 introduces a new syntax for calling the new operator, which is called *diamond* or simply "<>." It is used in place of a parametric type argument list such as <String, List<String>> when we instantiated contacts in the example above. The compiler interprets it as a point where the types shall be inferred from the context and usage.

Back to the declaration of strings of type List<String>, it is easy to infer that the type for a LinkedList to be assigned must resolve as LinkedList<String>. The code above can thus be rewritten as shown in **Listing 6**.

The change is marginal when a single parametric type is used, such as for strings, but it becomes much more interesting when more of them are involved, especially when they also require nested parametric types, such as for the declaration of contacts.

Diamond may be used with spaces between < and >, but this is discouraged:

```
List<String> strings =
new LinkedList<   >();
```

It must be noted that a type declaration involving diamond should not be confused with a raw type. Hence, an instance created with new LinkedList<>() does not have the same type as one created with new LinkedList().

As with all good things, certain restrictions limit the applicability of generic

LISTING 4 LISTING 5 LISTING 6

```
public static boolean isTrue(String s) {
  String str = s.trim().toUpperCase();
  int jump = -1;
  switch(str.hashCode()) {
    case 2404:
      if (str.equals("KO")) { jump = 3; }
      break;
    case 2497:
      if (str.equals("NO")) { jump = 4; }
      break;
    case 2524:
      if (str.equals("OK")) { jump = 0; }
      break;
    case 87751:
      if (str.equals("YES")) { jump = 1; }
      break;
    case 2583950:
      if (str.equals("TRUE")) { jump = 2; }
      break;
    case 66658563:
      if (str.equals("FALSE")) { jump = 5; }
  }
  switch(jump) {
    case 0:
    case 1:
    case 2:
      return true;
    case 3:
    case 4:
    case 5:
      return false;
    default:
      throw new IllegalArgumentException("Not a valid true/false string.");
  }
}
```

See all listings as text

type inference with diamond. Especially, diamond cannot be used on anonymous inner classes. Some developers enjoy initializing a collection, especially maps, using an anonymous inner class and an instance-initializing block such as in the following:

```
Map<String, String> map =
new HashMap<String, String>() {
  {
    put("foo", "bar");
    put("bar", "baz");
  }
};
```

We could be tempted to take advantage of diamond as follows:

```
Map<String, String> dmap =
new HashMap<>() {
  {
    put("foo", "bar");
    put("bar", "baz");
  }
};
```

But the code in **Listing 7** does not compile.

The updated Java Language Specification requires that the usage of diamond be forbidden for any anonymous class definition. This might seem surprising, especially given the previous example, where you would naturally infer HashMap<String, String>. The reason for this is related to the problem of *nondenotable* types. To understand this issue, let's consider the following class definition:

```
class SomeClass<T extends
Serializable & CharSequence> { }
```

This class has a parametric type T with a restriction that it must be a subtype of both Serializable and CharSequence. Internally, the compiler understands T as an *intersection type*, that is, a Serializable+CharSequence type. However, the intersection of two interfaces does not result in a runtime type that can be represented inside the Java Virtual Machine bytecode specifications. Hence, it is not a denotable type. The compiler performs generic type erasure and uses a type that does not violate the semantics of the program while being denotable. Consider this:

```
SomeClass<?> foo =
new SomeClass<String>();
```

This code compiles, because Serializable+CharSequence satisfies as a supertype for String. Now, we can even create an anonymous class from it:

```
SomeClass<?> fooInner =
new SomeClass<String>() { };
```

REDEFINED
Diamond cannot be used for any **anonymous** class definition.

You should not forget that each anonymous class declaration yields a genuine class declaration in the form of a .class file in the form EnclosingClass$1.class, EnclosingClass$2, and so on.

LISTING 7 / LISTING 8 / LISTING 9

```
Diamond.java:43: error: cannot infer type arguments for HashMap;
    Map<String, String> dmap = new HashMap<>() {
                                          ^
 reason: cannot use '<>' with anonymous inner classes
1 error
```

See all listings as text

We can check the content of such files with the javap decompiler that comes with the Java Development Kit, as shown in **Listing 8**.

This helps in understanding why anonymous inner classes are problematic when combined with diamond. Indeed, the following declaration is valid:

```
SomeClass<?> bar =
new SomeClass<>();
```

However, the declaration in **Listing 9** generates an error.

Let us put ourselves in the shoes of the compiler for a moment. We are requested to generate a class whose supertype is SomeClass<T extends Serializable & CharSequence>. We now need to infer a denotable type to be used instead of the nondenotable type Serializable+CharSequence, but clearly, the context gives us no clue for how to do that. To put it another way, we do not know of any denotable subtype to Serializable+CharSequence that could be used here. Given that Object is the closest reifiable type to be tested, but it is neither a CharSequence nor a Serializable, the compilation fails for a good reason.

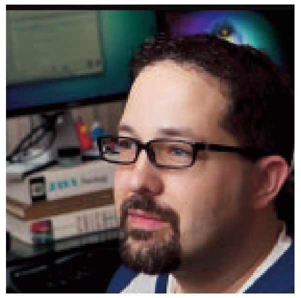
This change in the Java language was anticipated by many developers, and it should lead to an increased concision in source code. This should also reduce the apparent friction that some developers had with generics.

Conclusion

This article introduced three additions to the Java language in Java SE 7 that will certainly help you be more productive. In the next article, we will cover the remaining three items of Project Coin. </article>

LEARN MORE

- Project Coin mailing-list archives
- Project Coin blogs
- Latest maintenance review of the Java Language Specification
- Java SE 7 API
- "Language Designer's Notebook: Quantitative Language Design"



JOSH MARINACCI

BIO

PHOTOGRAPH BY
CHRIS PIETSCH/GETTY IMAGES

Using JLayer in Your Swing Applications

Creating lots of interesting effects is effortless in Java SE 7 thanks to the new JLayer component in Swing.

In 2005, I cowrote the O'Reilly book *Swing Hacks* with Chris Adamson. In this book, we detailed one hundred interesting ways to push Swing to its limits. Many of the hacks involved installing a custom repaint manager or otherwise manipulating how components were drawn to the screen. While they were a lot of fun, the hacks could be convoluted and dangerous. We had to work around bugs and eccentricities in Swing, including differences between the platforms. We also dug into private APIs that could and did change over time.

At the end of the day, we had a book full of hacks—fun code snippets that did interesting things but could break at any moment. They were great for playing around but not good enough for heavily used production code. Swing needed something better, and Java SE 7 has finally provided it: JLayer.

Java SE 7, many years in the making, has lots of new stuff under the hood but few new client-side APIs. One big exception, however, is the new JLayer Swing component.

This clever class is a tool for achieving all sorts of interesting effects with standard Swing code, and best of all, it doesn't require *any* hacks. Everything is built right in so you can quickly build interesting applications and *know* that they will work. JLayer started life as the open source JXLayer component, which was created by Alexander Potochkin, and it has since been added to the Java SE 7 core APIs.

NO HACKS NEEDED
JLayer is a tool for achieving all sorts of interesting effects with standard Swing code, with no hacking required.

At its most basic, JLayer is a Swing container. It holds regular Swing components and then manipulates their input and output. When the child component draws itself, JLayer can draw on top of the child, apply transforms, or redirect the child into a buffer for later manipulation.

You can also track and modify the mouse and keyboard inputs to the component. With these simple tools, you can create overlay indicators and transition effects, apply blurs and color blending, build animated spinners, add custom right-click menus, and monitor keystrokes.

Let's start with a few simple examples to see how it works.

Note: The source code for the examples described in this article can be downloaded here.

Example of Drawing on Top of a Control

Listing 1 shows how to draw on top of a control with transparent red. The code does three things:

- Creates a button
- Creates a LayerUI overlay that knows how to draw on top of buttons
- Creates a JLayer to combine the button and the overlay

The LayerUI subclass implements the paint(graphics,jcomponent) method. The call to super.paint(g,c) draws the button normally, and then the g.fillRect() call fills the component with translucent red (the "128" in the Color constructor is to make it 50 percent opaque). **Figure 1** shows transparent red drawn on top of a control.

Notice that the component to be drawn on is included as an argument to the paint method. This is *very* important. The LayerUI is typed to the component it will

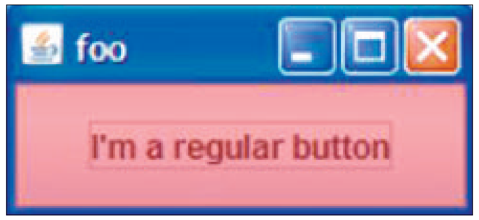


Figure 1

draw on (JButtons, in this case), but it doesn't store an internal reference to the component.

Instead, the current JLayer is passed in, and from the JLayer, you can get the wrapped component with getView(). This means the LayerUI is independent of the component it is drawn on top of and can be reused with *multiple* components. This important feature of the JLayer design lets you create reusable objects that can be applied throughout your application.

For example, if you are doing form validation and want to decorate any invalid form with a red overlay and an *x* graphic, you have to define this overlay only once. Then you can reuse the same instance on all your components. This clever design makes the JLayer and LayerUI far more flexible than previous skinning efforts.

POWERFUL SWING

JLayer is a very powerful addition to Swing that makes many previously hard or impossible effects easy to do in a clean and supported way.

Field Validator Example

Next, we will create a simple invalid field decoration. Any *invalid* field will be drawn over in red. Any *valid* field will be drawn over in blue.

The code in **Listing 2** is fairly simple. It creates a panel with a box layout containing three fields, each wrapped in a JLayer.

Listing 3 is the overlay.

The isValid function checks each field with a regular expression to see whether it's valid. In **Figure 2**, the zip code field is currently *not* valid because it contains a letter. If you replace the *x* with a number, the field will turn blue.

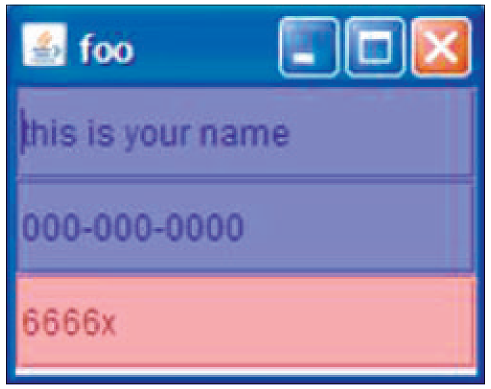


Figure 2

Event Filtering Example

The field validator works quite well. However, notice that the valid fields are clickable.

Let's suppose we want to let the user edit only invalid fields. We can block all mouse input to the valid fields using another feature of JLayer: input filtering.

Listing 4 shows some additional methods for filtering mouse input. Notice first that the LayerUI now sets and resets the LayerEventMask when the UI is installed and uninstalled. By default, the JLayer won't mess with mouse events at all because it would slow down the app.

To turn on event filtering, you have to set a layer mask when the LayerUI is installed in the JLayer, which is what the installUI and uninstallUI methods do. Then the code overrides the processMouseEvent method to filter the mouse events. It checks to see whether the current component is valid and consumes the event.

If the field isn't valid, it just calls super, which will block click events to valid fields. A more advanced version of this code could disable keyboard access as well.

You might wonder why the paint method of LayerUI passes in a JComponent instead of a JLayer. That's because LayerUI still has to implement the ComponentUI API that is part of Swing's Look and Feel system. ComponentUI defines the JComponent return type of installUI and uninstallUI. You can always cast this to a JLayer and then call getView to get the underlying component being decorated.

Real-World Examples

So far we've used JLayer for some trivial overdrawing effects, but there are many

LISTING 1 / LISTING 2 / LISTING 3 / LISTING 4

```
public class JLayerTest1 {
  public static void main(String[] args) {
    LayerUI<JButton> overlay = new LayerUI<JButton>() {
      @Override
      public void paint(Graphics g, JComponent clayer) {
        super.paint(g, clayer);
        g.setColor(new Color(255,0,0,128));
        g.fillRect(0,0,clayer.getWidth(),clayer.getHeight());
      }
    };

    JButton button = new JButton("I'm a regular button");
    JLayer<JButton> layer = new JLayer<JButton>(button, overlay);

    JFrame frame = new JFrame("foo");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.add(layer);
    frame.pack();
    frame.setVisible(true);
  }
}
```

See all listings as text

more-useful things you could do with it. Let's look at a few.

When trying to debug applications it's common to want to know the bounds of every component in the frame, for example, when one of your components seems to be invisible. While you could print this information to the console, with JLayer you can simply draw debugging information directly on top of the components.

The example shown in **Listing 5** and **Figure 3** draws the borders of every component in orange. It also writes the name of the component's class. This is handy if you want to tell the difference between many button subclasses.

GOING DEEP

The JLayer is created only once and set on the topmost panel. Because the **LayerUI** is recursive, it can draw on top of child **components as deep in the hierarchy** as necessary.

The code in **Listing 5** again creates a LayerUI instance with a paint method. This time it recursively goes through the component and its children to draw borders. It also writes out the simple name of the component's class if the component doesn't have any children. (It skips components with children to avoid overdraw on nested panels.)

Note that the JLayer is created only once and set on the topmost panel. Because the LayerUI is recursive, it can draw on top of child components as deep in the hierarchy as necessary.

The code in **Listing 6** is similar to what we've done before, but it draws any disabled component using a blur. That way, the user knows it is *really* disabled.

In this case, if the component is enabled, super() is called to draw normally. If the component is *not* enabled, the code creates an image buffer and calls super() on the graphics context from the image. This draws the component into the image instead of onto the screen. Then the code applies a simple blur operation to the image and draws

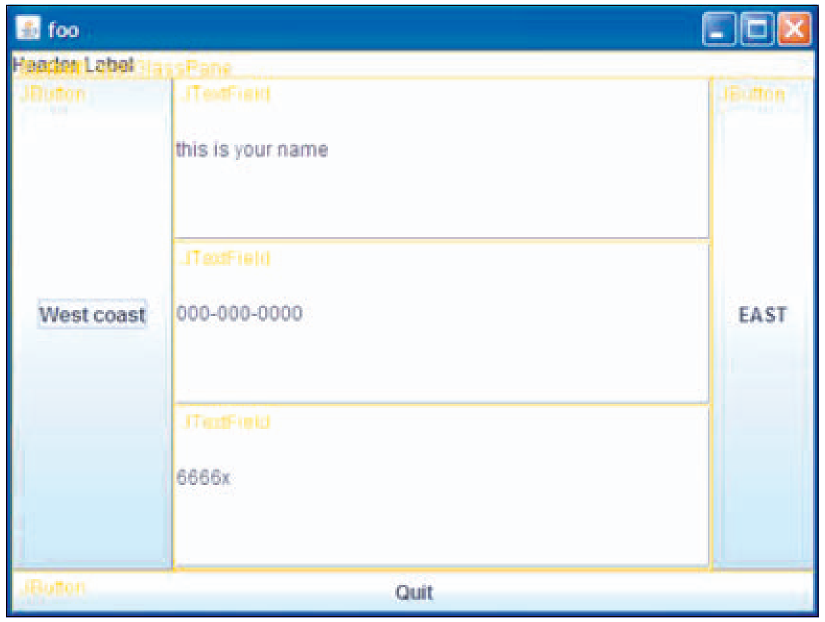


Figure 3

LISTING 5 LISTING 6

```
public static void main(String ... args) {
  LayerUI<JPanel> overlay = new LayerUI<JPanel>() {
    @Override
    public void paint(Graphics g, JComponent clayer) {
      super.paint(g, clayer);
      drawBorders(g,clayer);
    }
    private void drawBorders(Graphics g, JComponent clayer) {
      g.setColor(Color.ORANGE);
      g.drawRect(0, 0, clayer.getWidth(), clayer.getHeight());
      if(clayer.getComponentCount() <= 0) {
        g.drawString(clayer.getClass().getSimpleName(), 5, 15);
      }
      for(Component comp : clayer.getComponents()) {
        if(comp instanceof JComponent) {
          g.translate(comp.getX(),comp.getY());
          drawBorders(g,(JComponent)comp);
          g.translate(-comp.getX(),-comp.getY());
        }
      }
    }
  };
  JPanel panel = new JPanel();
  panel.setLayout( new BorderLayout());
  panel.add(new JLabel("Header Label"), BorderLayout.NORTH);
  panel.add(new JButton("Quit"),BorderLayout.SOUTH);
  panel.add(new JButton("EAST"), BorderLayout.EAST);
  panel.add(new JButton("West coast"), BorderLayout.WEST);
  JPanel subpanel = new JPanel();
  subpanel.setLayout(new BoxLayout(subpanel, BoxLayout.PAGE_AXIS));
  subpanel.add(new JTextField("this is your name"));
  subpanel.add(new JTextField("000-000-0000"));
  subpanel.add(new JTextField("6666x"));
  panel.add(subpanel,BorderLayout.CENTER);
  JFrame frame = new JFrame("foo");
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  frame.add(new JLayer(panel,overlay));
  frame.pack();
  frame.setVisible(true);
```

See all listings as text

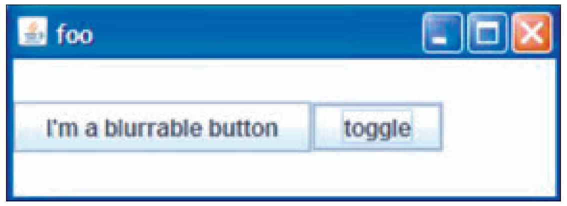


Figure 4

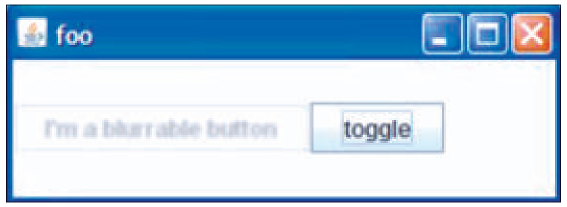


Figure 5

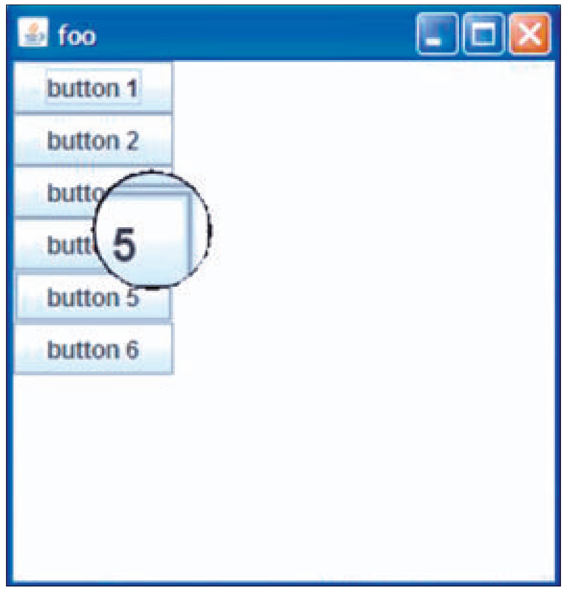


Figure 6

that final image to the screen. **Figure 4** shows an example of a button that can be blurred, and **Figure 5** shows a blurred button.

Because the JLayer repaints any time its view control changes, we don't need to track any state. A toggle button can simply toggle the enabled property of the button, and Swing takes care of the rest.

As a final example, **Listing 7** shows how you can do something completely crazy with JLayer by building a magnifying glass that will magnify whatever is underneath the mouse as the mouse moves, as shown in **Figure 6**.

The code shown in **Listing 7** is roughly the same as before. The difference is that this time it scales the graphics *before* drawing into an image. Then it draws the image back to the screen with a nice round clip.

Drawing components into a temporary image is a really useful technique because there are so many ways to post-process an image. To further extend the magnifying glass, you could add a distortion filter to give the scaled image a rounded effect.

Conclusion

JLayer is a very powerful addition to Swing that makes many previously hard or impossible effects easy to do in a clean and supported way. JLayer proves that Swing still has a lot of life left in it, and Java SE 7 provides new ways of extending Swing to do interesting things. (And being more generified is always nice, too.)

The examples shown here are just the tip of the iceberg. You could use JLayer to implement status overlays, busy indicators, global right-click menus, and much more. </article>

LEARN MORE

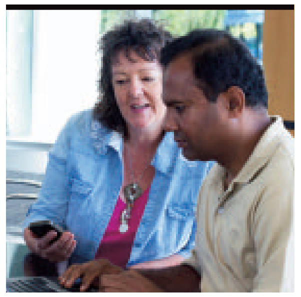
To learn more about **JLayer**, read the Java docs for **JLayer** and **LayerUI**:

- JLayer
- LayerUI

LISTING 7

```
    @Override
    public void paint(Graphics g, JComponent c) {
      //draw regular
      super.paint(g,c);
      //draw into buffer
      Graphics2D g2 = img.createGraphics();
      int ih = img.getHeight();
      int iw = img.getWidth();
      //fill with white
      g2.setPaint(Color.WHITE);
      g2.fillRect(0, 0, img.getWidth(), img.getHeight());
      //render view component scaled and translated
      Graphics2D g3 = (Graphics2D) g2.create();
      g3.translate(-pt.x + iw, -pt.y + ih*2);
      g3.scale(2, 2);
      g3.translate(-pt.x/2 - iw/4, -pt.y/2 - ih/4);
      super.paint(g3,c);
      g3.dispose();
      //draw black border
      g2.setPaint(Color.BLACK);
      g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.
VALUE_ANTIALIAS_ON);
      g2.setStroke(new BasicStroke(3));
      g2.draw(new Ellipse2D.Double(0,0,iw,ih));
      g2.drawRect(0,0, iw-1, ih-1);
      g2.dispose();
      //draw image to screen
      Shape oldClip = g.getClip();
      int mx = pt.x-img.getWidth()/2;
      int my = pt.y-img.getHeight();
      g.setClip(new Ellipse2D.Double(mx,my,iw,ih));
      g.drawImage(img, mx, my, null);
      g.setClip(oldClip);
    }
```

See all listings as text



WINSTON PRAKASH AND
SUSAN DUNCAN

BIO

PHOTOGRAPH BY BOB ADLER

Part 2

Understanding the Hudson Plug-in Development Framework

Add a UI component to configure a Hudson extension locally or globally.

This is Part 2 in a series of articles for beginners who are interested in understanding the fundamentals of Hudson plug-in development. Hudson is a popular open source continuous integration (CI) tool written purely in Java. Part 1 covered creating a Hudson plug-in using the HPI tool and creating custom implementations.

Part 2 covers adding, either locally or globally, the UI configuration elements a plug-in needs.

Configuration File Convention

Providing a configuration for a plug-in means providing some sort of UI with which the end user can configure your plug-in. In addition, the configuration UI provides feedback to the user about the validity of the input.

There are two ways to configure an extension. One is local to the area of the functionality the plug-in extends, and the other is through the Hudson-wide global configuration.

Hudson uses a UI technology called Jelly, a server-side rendering technology that uses a rendering engine to convert XML-based Jelly definitions (tags) to client-side code: HTML, JavaScript, and Ajax. Hudson provides a number of Jelly tags for your convenience.

A Hudson plug-in's model objects are bound to these tag attributes via Java Expression Language (JEXL). When the tags are rendered into HTML, JavaScript, and Ajax, the rendered code includes information from the model objects to which the attributes are bound. Using simple Jelly tags is a powerful way to express your UI.

The Jelly files used to render the UI have the extension .jelly. They reside in the resources directory of the plug-in. Hudson uses a heuristic convention to find these Jelly files. The folder under which these Jelly files must reside has a path hierarchy similar to the package name of the model class plus the name of the model class itself.

Hudson uses the namespace of the class package plus the model name as the folder hierarchy. For example, in Part 1 of this series, the model class HelloWorldBuilder has the package name org.sample .hudson. So the configuration file must reside in the following folder: org/sample/hudson/HelloWorldBuilder.

WHY JELLY?
Using **simple Jelly tags** is a powerful way to express your UI, rather than writing lots of HTML, JavaScript, and Ajax.

Local Configuration

Hudson uses another convention to tell whether the configuration file is meant for local configuration or global configuration. If the configuration file is named config.jelly, it is used as a local configuration file, and its content is included in the configuration of the functionality that this extension extends.

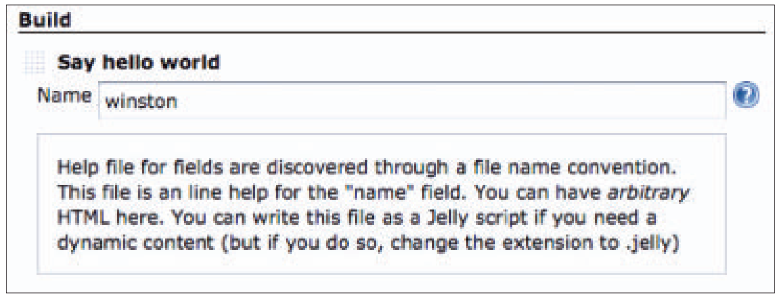
Because HelloWorldBuilder extends the Builder extension point, any Jelly content put in the configuration file org/sample/hudson/HelloWorldBuilder/config.jelly is included in the job configuration page to configure the HelloWorldBuilder extension in the builder section of the job configuration.

The HelloBuilder extension provides a UI for the user to configure. The UI provided by the HelloBuilder extension provides a simple text box for users to input their name.

Open and look at config.jelly by exploring the resource folder in the plug-in project. The content of the file is very simple (see **Listing 1**). It is a pure XML file with Jelly syntax.

There are two main tags playing the role of user interaction:

- entry tells Hudson that the enclosed tags are considered to be user interaction elements and they are submitted via an HTML form.



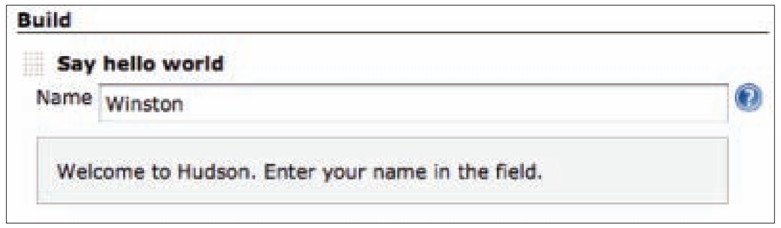
Build

Say hello world

Name winston

Help file for fields are discovered through a file name convention. This file is an line help for the "name" field. You can have *arbitrary* HTML here. You can write this file as a Jelly script if you need a dynamic content (but if you do so, change the extension to .jelly)

Figure 1



Build

Say hello world

Name Winston

Welcome to Hudson. Enter your name in the field.

Figure 2

- textbox renders a simple HTML text field whose value will be sent back to the server.

Understanding the UI Rendering

Let's take a closer look at how the UI is rendered from the Jelly file. In Part 1 of this series, we created a Hudson project called TestProject. Open the TestProject job configuration page by clicking the **Configure** link. Scroll down to the build section and view the Say Hello World builder and its configuration. Click the **Help** button ("?") on the right side of the text box. It displays online help text, as shown in **Figure 1**.

Let's find out where this help text comes from. The content of config .jelly has no such help text. Once again, convention comes into play. The folder containing the config file also has a file named help-name.html. Open the file using an editor and notice that it contains the exact help text shown in **Figure 1**.

How does Hudson know to get the content from this file and display it as help content for the field? The trick is in the name of the file. By convention, Hudson looks for a specific filename in the same folder that contains the config file. The name of the file should be help-{*fieldName*}.html.

In the config.xml file we have the following:

```
<f:entry title="Name"
field="name">
<f:textbox />
</f:entry>
```

field="name" indicates that the text box should be used as an entry field with the name Name. So, based on convention, the help text for that field should exist in a file named help-name.html.

The content of the help-name.html file is pure HTML. You can include images, text, and hyperlinks in the content to emphasize and enhance your help text. As mentioned in the example help text in **Figure 1**, if you want to use information from Hudson model objects, you should have Jelly content in the field help file and the file extension should be .jelly instead of .html. **Listing 2** shows an example file called help-name.jelly.

WHAT'S INSIDE?

The content of the help-name.html file is **pure HTML**; you can include images, text, and hyperlinks.

LISTING 1 LISTING 2

```
<j:jelly xmlns:j="jelly:core" xmlns:st="jelly:stapler" xmlns:d="jelly:define"
xmlns:l="/lib/layout" xmlns:t="/lib/hudson" xmlns:f="/lib/form">
<!--
Creates a text field that shows the value of the "name" property.
When submitted, it will be passed to the corresponding constructor parameter.
-->
<f:entry title="Name" field="name">
<f:textbox />
</f:entry>
</j:jelly>
```

See all listings as text

When Hudson is stopped and restarted with the above Jelly help file, clicking the Help button for the same text field displays the text shown in **Figure 2**.

Note that ${app.displayName} is replaced with Hudson, which is the name of the application.

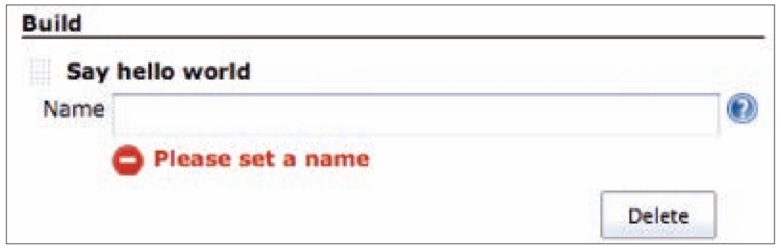
UI and Model Object Interaction

Now let's see how the UI interacts with Hudson model objects. HelloBuilder is a Hudson model object. It encapsulates data. The UI can interact with this model to get and display its data, get information from the user via fields in the UI, and update the model data.

The help file, help-name.jelly, contains a JEXL expression ${app .displayName}. When the server side of the Hudson application receives the request to render the job configuration page, it includes both the snippets config.jelly and help-name.jelly in the job configuration page, which itself is another Jelly file.

The entire Jelly file is given to the Jelly renderer to render into the client-side code. The Jelly renderer is responsible for substituting the corresponding value for the JEXL expression after evaluating the expression. The first part of the expression evaluates to the model object and then to the method name of the model object. By default, Hudson registers three identifiers for the model objects to the JEXL expression evaluator:

- app, which is the Hudson application itself. For example, ${app .displayName} evaluates to Hudson .getDisplayName().
- it, which is the model object to which the Jelly UI belongs. For example, ${it.name} evaluates to HelloWorldBuilder.getName().
- h, which is a global utility function called Functions that provides static utility methods. For example, ${h.clientLocale} evaluates to Functions.getClientLocale().



Build

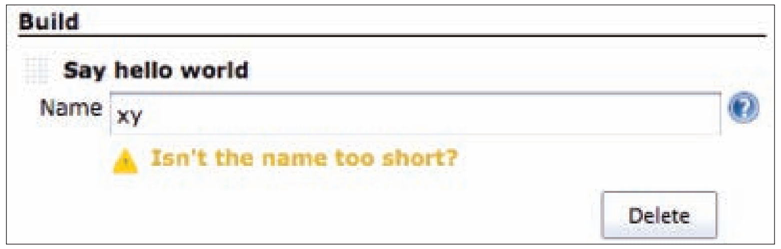
Say hello world

Name

Please set a name

Delete

Figure 3



Build

Say hello world

Name xy

Isn't the name too short?

Delete

Figure 4

Because the expression ${app.displayName} evaluates to Hudson, which is the name of the Hudson application, "Hudson" gets inserted into the field help text.

While the UI displays the data of a model, the input from the user in the UI updates the model data when the configuration page is submitted by the user. In this case, the value of the name the user enters in the UI ("winston") is updated in the model.

When the configuration page is submitted, Hudson re-creates the model by passing the corresponding value via the constructor. Hence, the constructor of the model object must have a parameter whose name matches the name of the field. In our configuration we have <f:entry title="Name" **field="name"**>.

So the constructor of your HelloBuilder must have a parameter named name. If you look at the constructor of the class HelloWorldBuilder, it has a name parameter, as follows:

```
@DataBoundConstructor
public HelloWorldBuilder
(String name) {
this.name = name;
}
```

The annotation @DataBoundConstructor hints to Hudson that this extension is bound to a field, and upon the UI submission, it must be reconstructed using the value of the submitted field.

Also the model must have a getter with the name of the field in the config.xml file to get the data for the second time around when the project is configured again, for example:

```
public String getName() {
return name;
}
```

This information is persisted along with the project configuration, as shown in **Listing 3**. Note that the value of the name field is saved as the HelloWorldBuilder configuration under the builders section.

UI Validation Methodology

In the job configuration page, under the build section of HelloBuilder, if you remove the value from the text field and click elsewhere on the page, a validation error message is displayed (see **Figure 3**).

LISTING 3

```
<?xml version='1.0' encoding='UTF-8'?>
<project>
 <actions/>
 <description></description>
 <keepDependencies>false</keepDependencies>
 <creationTime>1314407794225</creationTime>
 <properties>
    <watched-dependencies-property/>
 </properties>
 <scm class="hudson.scm.NullSCM"/>
  <advancedAffinityChooser>false</advancedAffinityChooser>
  <canRoam>true</canRoam>
 <disabled>false</disabled>
 <blockBuildWhenDownstreamBuilding>false</blockBuildWhenDownstreamBuilding>
 <blockBuildWhenUpstreamBuilding>false</blockBuildWhenUpstreamBuilding>
 <triggers class="vector"/>
 <concurrentBuild>false</concurrentBuild>
 <cleanWorkspaceRequired>false</cleanWorkspaceRequired>
 <builders>
    <org.sample.hudson.HelloWorldBuilder>
    <name>Winston</name>
    </org.sample.hudson.HelloWorldBuilder>
  </builders>
  <publishers/>
  <buildWrappers/>
</project>
```

See all listings as text

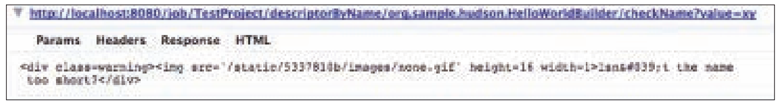
Note: Do *not* press the Enter (or Return) key. Doing so will submit the configuration page.

If you enter just a two-letter word (for example, xy) in the name field and click elsewhere, the information message shown in **Figure 4** is displayed.

Let's examine how Hudson displays this information message. Again, as seen in **Listing 3**, config.jelly does not include the messages. The magic is in the Jelly file rendering. Some Ajax code is rendered that, behind the scenes, contacts the Hudson server and asks what message should be displayed.

These Ajax requests are easily observable using Firefox and Firebug. When config.jelly is rendered by Hudson, the Jelly tag <f:textbox /> renders the Ajax code required to do the checking. Firebug displays the Ajax request info shown in **Figure 5**.

The Ajax request shown in **Listing 4** is sent to the Hudson server.

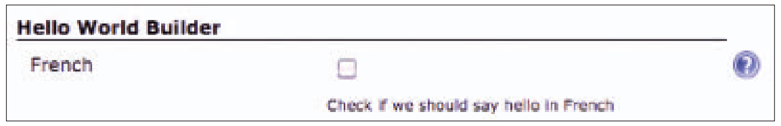


http://localhost:8080/job/TestProject/descriptorByName/org.sample.hudson.HelloWorldBuilder/checkName?value=xy

Params Headers Response HTML

```
<div class=warning><img src='/static/5337810b/images/none.gif' height=16 width=1>Isn&#039;t the name
too short?</div>
```

Figure 5



Hello World Builder

French ☐

Check if we should say hello in French

Figure 6

Hudson evaluates this request, finds the extension HelloWorldBuilder, executes the method checkName(), and returns the result. Again, Hudson uses the convention doCheck followed by *{nameOfTheField}* as part of the Ajax URL.
Note: By default, for every f:textbox tag in the Jelly config file, Hudson will render the Ajax check. However, if your extension class does not include the corresponding method (in this case, doCheckName()), Hudson will silently ignore the check request.

The doCheckName() method is straightforward. The Ajax request specifies the checkName method with the parameter value as {..}/checkName?value="xy". Hence, the parameter value of doCheckName must be annotated with the annotation @QueryParameter. Also, the method must return a FormValidation object, which determines the outcome of the check. See **Listing 5**.

The method doCheckName returns FormValidation.error(..) when an error occurs. The HTML sent back to the client displays the text in red along with an error icon (see **Figure 3**). If FormValidation.warning() is returned, the HTML sent back displays the message in yellow along with a warning icon (see **Figure 4**).

Global Configuration

Until now, we have concentrated on how to configure a plug-in at the job level. However, some of the configuration of an extension could be global. For example, if you use Git for source configuration management (SCM) in a project, you might want to configure two different jobs to force Git to check out from a different repository. So in both the projects, the Git SCM plug-in must be configured to use two different repository URLs.

However, if the Git SCM extension needs to know about the Git native binaries, placing the Git configuration UI that configures the Git binary location at the job level makes little sense, because it doesn't vary from project to project. It makes sense to put such a configuration in Hudson's global configuration page.

For Hudson to include the configuration of a plug-in in its global configuration page, the configuration must be declared in a file called global.jelly. The namespace convention for this file is similar to the convention for local configuration. For the HelloWorldBuilder global config file, it is org/sample/hudson/HelloWorldBuilder/global.jelly.

In Part 1 of this series, we saw that the HelloWorldBuilder.perform(..) method includes the code shown in **Listing 6**.

HelloWorldBuilder can be configured to say hello in either French or English. The configuration of which language to use is set globally. Once set, all the builds of various jobs that use HelloWorldBuilder would say hello in either French or English.

The global configuration of the HelloWorldBuilder plug-in is done in the Hudson configuration page in the Hello World Builder section (see **Figure 6**).

Here, the user sets the global configuration to control whether French is used as the language. In global.jelly, the checkbox is defined as shown in **Listing 7**.

When the user edits the Hudson configuration page, the decision about whether this checkbox should be selected comes from the extension itself. The JEXL expression ${descriptor.useFrench()} would resolve to HelloBuilder.DescriptorImpl.useFrench(), which is defined as follows:

```
public boolean useFrench() {
return useFrench;
}
```

useFrench is a field in HelloWorldBuilder. This field is set to true if the user selects the checkbox. Once the global configuration is submitted, by convention, Hudson calls the method HelloBuilder.DescriptorImpl.configure() and passes a JSON object corresponding to the page submission. It is up to the extension to find its submitted value and then apply it. HelloWorldBuilder defines this method as shown in **Listing 8**.

In this method, the Boolean value of the field useFrench is obtained and used to set HelloWorldBuilder.useFrench. The next time HelloBuilder.perform() is called during the build of the job, HelloBuilder.useFrench is consulted and based on its value, the hello message is written in either French or English.

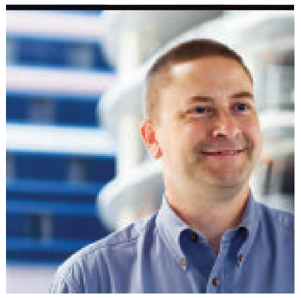
Conclusion

The Hudson plug-in development environment provides a rich set of extension points with which plug-in developers can develop custom plug-ins. In this article, we explored ways to configure the extensions in a plug-in using the Jelly UI framework . </article>

LISTING 4 / LISTING 5 / LISTING 6 / LISTING 7 / LISTING 8

```
GET /job/TestProject/descriptorByName/org.sample.hudson.HelloWorldBuilder/
checkName?value=xy HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.6; rv:6.0.1) Gecko/20100101
Firefox/6.0.1
Accept: text/javascript, text/html, application/xml, text/xml, */*
```

See all listings as text



SIMON RITTER

BIO

PHOTOGRAPH BY BOB ADLER

Part 2

JavaFX and Swing Integration

Migrate a Swing application to use the richness of JavaFX without completely rewriting the application.

In Part 1 of this three-part series, we looked at how to use the JFXPanel class to wrap a JavaFX scene into a component that could be added to a Swing Container.

In this article, we'll build on that to show how we can migrate a Swing application to use the richness of JavaFX without having to rewrite it completely. The sample Swing application will be modified to replace the table component to show how we can use some of the exciting functionality of JavaFX to improve the look and feel of the application.

CSS SUPPORT
An exciting **feature of the JavaFX platform** is support for cascading style sheets (CSS).

Structure of the Sample Application

The sample application, StocksMonitor, provides a Swing-based interface to allow a number of stock prices to be tracked. The code for this application can be downloaded here.

The application has been well organized into packages, allowing us to easily identify the relevant parts. For our purposes, we'll be modifying the StocksMonitorMainWindow and Quote classes and replacing the QuoteTable class with a new QuoteFXTable class.

For the sake of simplicity, the aim will be to make the QuoteFXTable class fully compatible with the existing QuoteTable class. To do this, we will need to implement the Observer and PropertyChangeListener interfaces and include a setQuoteTable method with the correct signature.

JavaFX TableView Control

The JavaFX API provides the TableView control, which is designed to visualize an unlimited number of rows of data, broken out into columns. TableView provides the TableColumn API, which supports features such as cell factories for customizing cell contents, column reordering by the user at runtime, and column nesting to group sets of columns.

Using a TableView control is similar in design to the JTable component in Swing. The model-view-controller pattern is used, so a data model needs to be constructed that can be passed to the TableView so it can be displayed.

In the original application, the data model is encapsulated in the Quote class. To enable us to use this as the model for a TableView, we need to add support for JavaFX binding, which uses JavaFX properties. For each column that we want to display in our table, we need a property of the appropriate type, an accessor method for the property, and a method to call when the values are changed.

Stephin Chin talks about JavaFX with *Java Magazine*'s Tori Wieldt at Devoxx 2011 .

Table Data Model

Listing 1 shows the changes that need to be made to the Quote class.

For the name of the stock, we create a SimpleStringProperty, which is a concrete implementation of the abstract StringProperty class. This property is then associated with the name value of the Stock object.

For the other values, we create SimpleFloatProperty objects. Note that the accessor methods don't follow the convention of *getNameOfProperty*. They simply make the method name the same as that of the property instance.

The updateValues method simply sets the values of the individual properties using the existing methods and references in the Quote class.

TableView Node as a Swing Component

Now that we have the appropriate data model we can construct our TableView and wrap it using JFXPanel. This requires two classes, one to act as the Swing component (QuoteFXTable) and one for the TableView node (QuoteFXTableNode).

Listing 2 shows the QuoteFXTable class.

To be compatible with the QuoteTable class, we implement the PropertyChangeListener and Observer interfaces and extend JFXPanel, which is a subclass of JComponent. The constructor simply has to call the constructor of the superclass, instantiate a new QuoteFXTableNode object, and add a Runnable task to the JavaFX application thread to set the scene graph.

GET VISUAL
The JavaFX API provides the **TableView** control, which is designed to visualize an unlimited number of rows of data, broken out into columns.

Notice that we do not need to concern ourselves with the size of the scene or table, because this is handled automatically by the Swing layout manager and JavaFX scene graph. All the required methods pass the call through to the QuoteFXTableNode class where the real work happens.

Listing 3 shows part of the QuoteFXTableNode class.

For each column that we want in the table, we need to instantiate a TableColumn object. We use generic type parameters to define the type of the TableView (Quote) and the type of the data to be displayed in this column (a string for the name of the stock and float for the numeric values).

The cell value factory needs to be set to specify how to populate the cells within a single TableColumn. For this common case, where we are populating the cells using a single value from a Java bean, we use the PropertyValueFactory class. The constructor for this takes the name of the property that contains the data for this column (which will match up to the property in the Quote class). Having defined all the columns, we add these to the table using the setAll method.

The implementation of the setQuoteTable method is trivial. We clear and set the list of quotes to be displayed

LISTING 1 LISTING 2 LISTING 3

```
public Quote(Stock aStock, BigDecimal aCurrentPrice, BigDecimal aChange) {
  fStock = aStock;
  fCurrentPrice = aCurrentPrice;
  fChange = aChange;
  stockProperty =
    new SimpleStringProperty(this, "stock", fStock.getName());
  priceProperty =
    new SimpleFloatProperty(this, "price");
  changeProperty =
    new SimpleFloatProperty(this, "change");
  percentChangeProperty =
    new SimpleFloatProperty(this, "percentChange");
  profitProperty =
    new SimpleFloatProperty(this, "profit");
  percentProfitProperty =
    new SimpleFloatProperty(this, "percentProfit");

  validateState();
 }
...
 /* New methods required for JavaFX table */
 public void updateValues() {
  setPrice(fCurrentPrice.floatValue());
  setChange(fChange.floatValue());
  setPercentChange(getPercentChange().floatValue());
  setProfit(getProfit().floatValue());
  setPercentProfit(getPercentProfit().floatValue());
 }
 /**
  * @return the stockProperty
  */
 public StringProperty stockProperty() {
  return stockProperty;
 }
 /**
  * @param stock The stock name
  */
 public void setStock(String stock) {
  stockProperty.set(stock);
 }
```

See all listings as text

and then call the update method for each quote. This handles changes to both stock prices as well as to the portfolio, such as adding or deleting stocks.

The only other change required to complete the integration of the TableView is to modify the StocksMonitorMainWindow class to replace the QuoteTable reference and instantiation with QuoteFXTable.

Let's Make It Interesting

Right now, all we've done is replace the JTable component with a TableView node. That's nice, but it doesn't make our UI significantly better. Let's use some more JavaFX code to make the display more interesting.

FEEL THE POWER

Modifying the style sheet will change the **look and feel** of the application without the need to change the code. **Style sheets are very powerful** and can be used to completely change the look of an application when there are numerous JavaFX nodes.

For a stock price monitor, it would be good if we could highlight when a change happens: red for a decrease and green for an increase. To make this visually interesting and to attract the user's eye, we'll have the price flash on the screen. The number will fade in and out to give it a "softer" feel.

To do this, we'll need to provide our own customized TableCell class to render the cells of the price column.

Listing 4 and **Listing 5** show the code for the QuoteFXPriceCell class.

If you refer back to the code in **Listing 3** for the QuoteFXTableNode class, you will see that in addition to setting the cell value factory for the price column, we also set the cell factory to a Callback object. All the call method needs to do is return a new instance of the QuoteFXPriceCell class. We pass it a reference to a list of quotes, which we'll need to determine the change in price.

To change the way the cell is rendered, we override the updateItem method. For everything to work properly, it is important that we call the overridden method at the start. Because null is a valid value for a cell, we check the empty flag and do nothing if the flag is true. We also return if the value is null, because this value has no meaning to us in this context. The text for the cell is set using the value passed as an argument.

TableCell extends from the Labeled class, which is common to all controls that have a text label (such as a button). This also provides support for including a graphical icon with the text of the label. This icon is a node, so it can be a complete scene graph, if required. The QuoteCellGraphic class provides a node, which graphically shows whether a price has gone up, gone down, or is unchanged by using a colored triangle or rectangle.

We can reference into the list of quotes that we got in the constructor using getIndex from the IndexedCell interface. Once we know how the price has changed, we can make the appropriate icon from QuoteCellGraphic visible.

LISTING 4 LISTING 5

```
@Override
 protected void updateItem(Float value, boolean empty) {
  super.updateItem(value, empty);

  if (empty)
   return;

  setText("" + value);
  icon = new QuoteCellGraphic();
  setGraphic(icon);

  float change = quotes.get(getIndex()).getChange().floatValue();

  if (change  > 0.0)
   icon.setUp();
  else if (change < 0.0)
   icon.setDown();
  else icon.setNoChange();

  SimpleDoubleProperty faderProperty = new SimpleDoubleProperty();
  opacityProperty().bind(faderProperty);

  Timeline faderTimeline = new Timeline();
  faderTimeline.getKeyFrames().add(new KeyFrame(Duration.ZERO,
    new KeyValue(faderProperty, 1.0)));
  faderTimeline.getKeyFrames().add(new KeyFrame(new Duration(500),
    new KeyValue(faderProperty, 0.1)));
  faderTimeline.getKeyFrames().add(new KeyFrame(new Duration(1000),
    new KeyValue(faderProperty, 1.0)));
  faderTimeline.setCycleCount(3);
  faderTimeline.playFromStart();
 }
}
```

See all listings as text

To make the price flash, we create a Timeline, which varies the opacity of the cell from 1.0 to 0.1 and back again over a period of one second. The cell is made to flash three times by setting the cycleCount appropriately.

Skinning

Another exciting feature of the JavaFX platform is support for cascading style sheets (CSS). Many nodes implement support for CSS, including TableCell.

Adding a style sheet to an application is pretty simple. If you refer back to the QuoteFXTable class in **Listing 2**, you will see that after we create the scene in our Runnable task, we add a reference to a CSS file to the style sheet list of the scene. This references a CSS file that we include in the same package as the QuoteFXTable class.

Now look at the QuoteFXTableNode class again in **Listing 3**, specifically the call method of the Callback class that we pass to the setCellFactory method of the price column. Here, we create a new QuoteFXPriceCell object and add to the style class list a reference to the name we use in our CSS file to set parameters.

The CSS file is shown below:

```
.fx-table:filled {
  -fx-background-color:
lightblue;
  -fx-border-color: black;
}
```

GET RICH

You can **migrate a Swing application** to use the richness of JavaFX without having to engage in a complete rewrite.

This is very simple. We create a style that has the name we used in the Java code and specify a pseudo-class that means this style will be used only on price cells that are filled. Two parameters are specified: one to set the color the cell is filled with and one to set the color of the border.

Modifying the style sheet will change the look and feel of the application without the need to change the code. Style sheets are very powerful and can be used to completely change the look of an application when there are numerous JavaFX nodes.

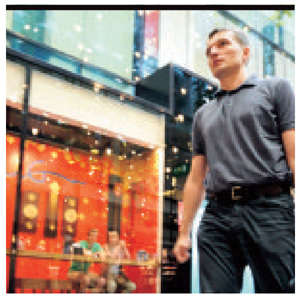
Conclusion

In this article, we've seen how we can migrate a Swing application to use the richness of JavaFX without having to engage in a complete rewrite. We've modified a sample Swing application to replace its table component in order to make use of the exciting functionality of JavaFX to improve the application's look and feel.

In Part 3, we'll look at ways to replace one of the configuration menus in our sample application. </article>

LEARN MORE

- JavaFX documentation and tutorials
- JavaFX 2.0 API documentation (Javadoc)
- JavaFX showcase



Lightweight Publish-Subscribe Communication

Java EE 6 and CDI can be used for lean, efficient implementation of local publish-subscribe communication.

ADAM BIEN

BIO

PHOTOGRAPH BY THOMAS EINBERGER/ GETTY IMAGES

The ancient—in internet time—J2EE specification used Java Message Service (JMS) as a workaround for asynchronous invocation of synchronous services as well as for the distribution of local events. JMS, unchanged in Java EE 6, has a verbose API.

With Contexts and Dependency Injection (CDI) 1.0 (JSR-299) and Enterprise JavaBeans (EJB) 3.1 (JSR-318), the J2EE workarounds and patterns aren't necessary. JMS is still essential for true messaging, but it is no longer required for the implementation of asynchronous processing or the implementation of a local Observer pattern.

Asynchronous Invocation

As described in the "Context" section of the Service Activator pattern, "Enterprise beans and other business services need a way to be activated asynchronously." The Service Activator pattern was the only way for EJB 2.x components to be "legally" invoked asynchronously. Service Activator can be considered a decorator that adds asynchronous nature as a cross-cutting concern to an already existing EJB interface.

The decoration usually comprised a JMS queue with a javax.jms.MessageProducer as sender and a message-driven bean as an asynchronous receiver. Even a Java EE 6 implementation of Service Activator is not lean. You need to implement a transactional message sender, which uses a javax.jms.Queue and a javax.jms.ConnectionFactory, to create and send a javax.jms.Message instance. Java EE 6 helps you only by the acquisition of the resources with Dependency Injection (JSR-330).

The major parts of the code in **Listing 1**, which shows javax.jms.MessageProducer wrapped with an EJB 3.1 bean, were generated with NetBeans 7 (using Alt+Insert on a Microsoft Windows system or Ctrl+I on a Mac system). However, the generated source code for the whole lifecycle of the project must be maintained.

A message-driven bean on the other end of the queue receives the message, casts it to a javax.jms.TextMessage, extracts its payload, and passes it to the synchronous service. **Listing 2** shows the asynchronous receiver.

The message-driven bean in **Listing 2** is used only for asynchronous invocation of the synchronous EJB bean. None of the JMS features or qualities are used. We are interested only in a nonblocking invocation by a thread from the application server's pool.

KEEP IT SIMPLE

The **main goals of JMS 2.0** will be simplicity and tight integration with the Java EE 7 specifications, especially with CDI.

Listing 3 shows the actual business logic. After the injection of the SynchronousService EJB 3.1 bean into ServiceActivator, the method message(String message) is decorated with the asynchronous onMessage method in the message-driven bean.

Note: The implementation described here is already simplified. In the original J2EE implementation, Dependency Injection wasn't available, so EJB 2 components also required the use of Java Naming and Directory Interface (JNDI) lookups to get access to other components. The implementation of the local EJBHome and local interfaces, as well as the existence of standard and proprietary XML deployment descriptors, was also necessary. Service Activator was a complex solution to a trivial challenge.

The Java EE 6 Answer

Java EE 6 made the implementation of the Service Activator pattern obsolete. **Listing 4** shows the

Service Activator equivalent in Java EE 6.

The entire JMS overhead can be replaced with a single javax.ejb.Asynchronous annotation. The semantics of both approaches, however, are not identical. In the case of Service Activator, you could use persistent queues to make the invocation more robust. The application server would redeliver a message until the processing succeeds.

In the @Asynchronous case, there is only one attempt. If it fails, the transaction will be rolled back and you would need to implement any recovery by yourself. On the other hand, java.util.concurrent.Future makes it easy for an EJB 3.1 bean to return a pointer to the return value. **Listing 5** shows the "fire and forget" approach with its result.

Service Activator was primarily designed for unidirectional fire-and-forget communication. In order to return the value of a Service Activator implementation, you need to create and send a temporary queue with the message and register another message-driven bean as "result listener" at the temporary queue end. The Java EE 6 future-based solution is orders of magnitude leaner than the J2EE "best practice."

JMS destinations are remote by convention. Although most of the JMS providers offer local queues as well, don't rely on that. @Asynchronous methods, on the other hand, are usually local, although it is also possible to use @Remote interfaces for remote communication.

NOT NECESSARY
Java EE 6 made the implementation of the Service Activator pattern obsolete.

Also, the JMS implementation is not type-safe. You need to wrap the actual payload into an appropriate javax.jms.Message (for example, a string in a javax.jms.TextMessage) at the sender side and unwrap it in the message-driven bean. The unwrapping requires you to check the type multiple times and cast the payload into the desired type. CDI "eventing," on the other hand, is type-safe. The payload arrives in the message listener "as is." No wrapping and unwrapping is required.

Although the spirit of Service Activator still exists, the entire JMS-related implementation is obsolete in Java EE 6. You can usually replace the vast majority of all Service Activators in J2EE with a single @Asynchronous annotation in Java EE 6 for local event distribution.

Publish-Subscribe: The Principle

EJB 3.1 made the Service Activator pattern obsolete. The CDI specification, on the other hand, can be used for lean and efficient implementation of local publish-subscribe communication. The publish-subscribe pattern (or pub/sub) is defined on Wikipedia as "a messaging pattern where senders of messages, called publishers, do not program the messages to be sent directly to specific receivers, called subscribers. Published messages are characterized into classes, without knowledge of what, if any, subscribers there may be. Subscribers

LISTING 1 / LISTING 2 / LISTING 3 / LISTING 4 / LISTING 5

```
@Stateless
public class ServiceClient {
  @Resource(mappedName = "jms/serviceactivator")
  private Queue serviceactivator;
  @Resource(mappedName = "jms/serviceactivatorFactory")
  private ConnectionFactory serviceactivatorFactory;

  public void broadcast(String message){
    Connection connection = null;
    Session session = null;
    try {
      connection = serviceactivatorFactory.createConnection();
      session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
      MessageProducer messageProducer = session.createProducer(serviceactivator);
      messageProducer.send(createTextMessage(session, message));
    }catch(Exception e){
      throw new EJBException("Cannot send message: " +e,e);
    } finally {
      if (session != null) {
        try {
          session.close();
        } catch (JMSException e) {
          /*...*/
        }
      }
      if (connection != null) {
        try {
          connection.close();
        } catch (JMSException ex) {
          /*...*/
        }
      }
    }
  }
  private Message createTextMessage(Session session, String messageData) throws
JMSException {
    TextMessage tm = session.createTextMessage();
    tm.setText(messageData);
    return tm;
  }
}
```

See all listings as text

express interest in one or more classes, and only receive messages that are of interest, without knowledge of what, if any, publishers there are. This pattern provides greater network scalability and a more dynamic network topology."

The key point of publish-subscribe is its dynamic nature. Listeners and broadcasters can come and go. There is a dynamic N:M relation between listeners and broadcasters; there is no requirement for the existence of any. A broadcaster could send a message without the existence of any listener. There could be several listeners without any broadcaster. The "message" in the publish-subscribe definition does not, however, have to be a javax.jms.Message. It is better to think about messages as events or payloads. With CDI and EJB 3.1, you can implement lean publish-subscribe communication without JMS.

WRITE MORE CODE
Although a **JMS publish-subscribe implementation** is more powerful and flexible, it also requires that significantly more code be written.

Java EE 6 (and CDI, in particular) comes with a built-in event: the injectable javax.enterprise.event.Event. It allows you to fire (distribute) any object you like to all listeners locally. If there is no listener, the broadcasted payload just disappears. To send a message you only have to inject the Event class and decide which kind of payload you would like to distribute. **Listing 6** shows a CDI String broadcaster.

The message is sent with a single invocation of the fire method. CDI events are transaction-aware and EJB 3.1 beans are transactional by convention. Because there is no configuration (annotation or XML), the broadcast method is going to be executed within a transaction.

The listener is required to implement only a void method with a single parameter annotated with the @Observes annotation. **Listing 7** shows a CDI String listener. The type of the parameter denoted with the @Observes annotation must match the type of the fired payload. Only then does the message get delivered; otherwise, it just disappears.

Unlike JMS, with CDI you can easily observe successful and unsuccessful transactions with a single listener at the same time. You only need to set the during element of the @Observes annotation. **Listing 8** shows an example of how to listen to commits and rollbacks at the same time. The message gets delivered only in the specified transaction phase. The @Observes(during=) element makes CDI events particularly useful for the implementation of batch job monitoring or audits. You can easily track all successful and failed transactions.

What About Multiple Destinations?

CDI eventing is type-safe. If the "fired" and "observed" types match, the event gets delivered; otherwise, it is ignored. With the current approach, you can use a type only once. There is no way to distinguish between important and tenuous messages. You could, of course, extend the type of the message by wrapping the string with a custom event class. **Listing 9** shows a custom event class for event dispatching.

LISTING 6 / LISTING 7 / LISTING 8 / LISTING 9 / LISTING 10

```
@Stateless
public class MessageBroadcaster {
  @Inject
  Event<String> event;
  public void broadcast(String message){
    event.fire(message);
  }
}
```

See all listings as text

However, introducing types is a poor approach for mimicking destinations. It results in class explosion and unnecessary bloat. CDI provides an elegant solution with qualifiers. **Listing 10** shows a qualifier for message dispatching. A qualifier is an annotation denoted with the @Qualifier interface. @Retention should always be set to RUNTIME, and with @Target, you need to choose the elements to which you would like to apply the annotation. The @Qualifier annotation can be considered to be a type extension. For matching, the CDI framework considers not only the types but also the applied qualifiers. You need to use the qualifier only at the injection point:

```
@Inject @Importance
(Importance.Degree.HIGH)
Event<String> event;
```

Then, annotate the corresponding parameter:

```
public void onImportantMessage
(@Observes @Importance
(Importance.Degree.HIGH)
String message){}.
```

There is still one deficiency: You need to inject an Event for every channel with the given qualifier. Instead of injecting the Event with different qualifiers multiple times, you can select the Qualifier on the fly, as shown in **Listing 11**, which shows dynamic channel selection. For the on-the-fly selection of the message listener, you only have to pass a Qualifier instance to the select method. The problem is that it is impossible to instantiate an annotation. **Listing 12** shows a helper class for annotation instantiation.

The class ImportanceSelector in **Listing 12** implements the Importance annotation and returns its enum value as well as the annotation type. The enum Degree is passed as a constructor parameter, and it can be used easily for the selection of the desired message type (see **Listing 11**).

Synchronous or Asynchronous

CDI events are delivered synchronously. The sender has to wait until the listener method completes. A RuntimeException inside the message body is propagated to the message sender. If the message sender is an EJB 3.1 bean (or at least in the invocation call stack), the transaction is automatically rolled back. This is convenient in cases where the sender would like to be decoupled from the listener but perform multiple tasks in one transaction consistently. Documenting the state of the business process by writing audits to a database with CDI events is an example of synchronous, transactional events.

Delivering events asynchronously is the natural way of publish-subscribe communication. The sender does not need to block until the event is delivered and processed. Also, JMS works asynchronously.

Events can be asynchronously consumed with a no-interface EJB 3.1 bean. You need to declare only the message listener as a @Stateless session bean and an @Asynchronous method. **Listing 13** shows asynchronous event consumption in an EJB 3.1 bean.

The @Asynchronous annotation also changes the transactional behavior. The method onImportantMessage is executed in a new, independent transaction. An exception thrown in the @Asynchronous onImportantMessage method will not cause the transaction initiated in the MessageBroadcaster EJB bean to roll back. This behavior is identical to a message processed by a message-driven bean bound to a javax.jms.Destination.

When Lightweight Is Not Enough

Asynchronous CDI events are similar to JMS topics without the availability of durable subscribers. Messages are stored persistently for a durable subscriber while it is inactive. On the next sign-in, all messages are redelivered to the subscriber.

A CDI event can be cached transiently in the thread pool's queue, and it can get lost after an application server crash or a transaction rollback caused, for example, by RuntimeException. Neither CDI nor @Asynchronous EJB 3.1 beans offer durable subscribers out of the box.

Also, event redelivery is not available in CDI. In contrast to JMS, the repetition of a failed transaction needs to be implemented by the application, not the infrastructure.

Although a JMS publish-subscribe implementation is more powerful and flexible, it also requires that significantly more code be written. For the implementation of an Observer pattern, CDI is well suited. Events are usually transient and do not require persistence or redelivery. A local CDI event distribution can be implemented with a fraction of the code required for adequate JMS functionality.

Real Messaging and JMS Strikes Back

CDI eventing with the EJB 3.1 bean @Asynchronous combination makes the Service Activator pattern obsolete. Although it looks tempting to entirely

LISTING 11 / LISTING 12 / LISTING 13 / LISTING 14

```
@Stateless
public class MessageBroadcaster {
  @Inject
  Event<String> event;
  public void broadcast(String message){
    event.select(new ImportanceSelector(Importance.Degree.LOW)).fire(message);
  }
}
```

See all listings as text

replace JMS with CDI eventing, there is only a small overlap between JMS and CDI. Only JMS provides you with "once and only once" delivery semantics. To meet the "once and only once" delivery quality you have to set up a persistent (backed by a database) queue. In the "sending" transaction, the message is persisted in the database first and delivered in subsequent transactions. Also in a publish-subscribe scenario the messages can be stored on the server during the subscriber's offline periods.

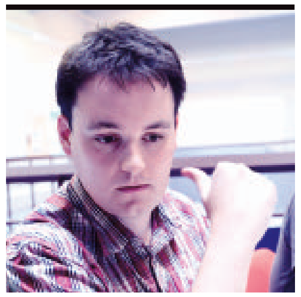
Neither use case above can be implemented with CDI events. CDI events are not persistent and would require you to implement "once and only once" delivery and, thus, to reinvent the wheel.

The old JMS specification is planned to be completely refurbished and shipped with Java EE 7. The main goals of JMS 2.0 will be simplicity and tight integration with the Java EE 7 specifications, especially with CDI. **Listing 14** shows a preview of JMS 2.0 (this could change at any time) that is as lightweight as CDI.

JMS 2.0 should eliminate the plumbing required to send and receive messages. The JMS 2.0 programming model is planned to be similar to CDI (see **Listing 14**). Even with JMS 2.0, you should not misuse messaging just to execute synchronous services asynchronously. On the other hand, CDI is perfect for transient event delivery, but it is really bad for messaging. Use the right tool for the job. </article>

LEARN MORE

- LightweightPublishSubscribe project
- *Real World Java EE Night Hacks—Dissecting the Business Tier* (2011)



JULIEN PONGE

BIO

Part 1

Clustering and High Availability Made Simple with GlassFish

The clustering features of GlassFish provide the ability to scale applications horizontally by running a cluster comprising multiple servers.

GlassFish is known for being the reference implementation of the Java EE specifications. It is also known for being a speedy modular server that is particularly well suited for development tasks. The capabilities of GlassFish do not stop after the development phase, though. GlassFish is also a production-class lightweight application server for apps requiring the most up-to-date implementation of Java EE.

Note: Two editions of GlassFish exist: GlassFish Server Open Source Edition and Oracle GlassFish Server. This article is relevant to both.

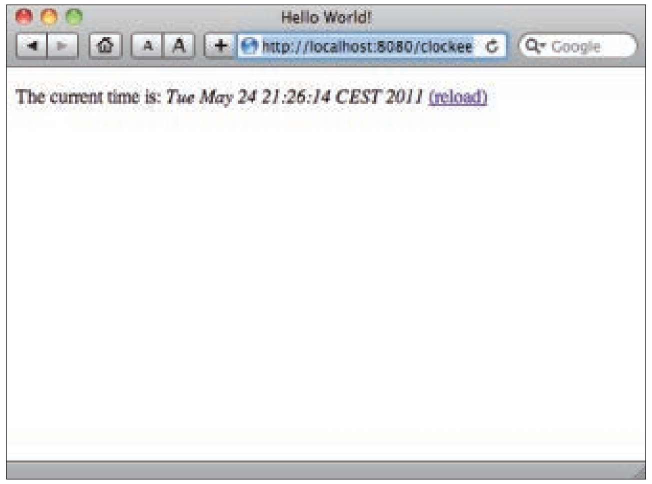
Two orthogonal problems arise in production: providing high availability of applications and absorbing traffic load. The solution lies in scaling applications horizontally by running not just one server but a *cluster* comprising multiple servers. High availability is achieved when instances running in a cluster can replicate session data so that a few instances joining and leaving a cluster do not cause applications to become unavailable.

This article is Part 1 of a two-part series on the clustering features of GlassFish and introduces centralized cluster provisioning and management of GlassFish servers. The second article discusses enabling high availability of applications in GlassFish clusters.

Note: The source code for the examples described in this article can be downloaded here.

Julien Ponge provides a brief introduction to his GlassFish series.

PHOTOGRAPH BY MATT BOSTOCK/GETTY IMAGES



Hello World!

http://localhost:8080/clockee

The current time is: Tue May 24 21:26:14 CEST 2011 (reload)

Figure 1

Meet ClockEE, Our Running Example

The running example that we will use in this article is deliberately simple to focus on the management of GlassFish clusters. It is a Web application called ClockEE that provides a single page on which the current time is displayed (see **Figure 1**).

The application is a Maven project whose source code is in the ClockEE.zip file. The sole view is provided using JavaServer Faces, as shown in **Listing 1**.

In turn, it requires the presence of a Contexts and Dependency Injection (CDI)–managed bean that provides the clock value, as shown in **Listing 2**.

The @Named annotation

makes instances of this bean available with CDI under the clock name, while the @ApplicationScoped annotation ensures that a single instance will be created for the whole application. The #{clock.now} expression in JavaServer Faces thus resolves to that bean and its getNow() method.

Installing GlassFish and Deploying ClockEE

As mentioned earlier, two editions of GlassFish exist: GlassFish Server Open Source Edition and Oracle GlassFish Server. The latter is a commercial distribution supported by Oracle and contains a set of extensions called Oracle GlassFish Server Control that focuses on performance, monitoring, and security. The two editions share the same code base, including clustering and centralized administration. The Open Source Edition is no lesser an edition.

GlassFish Server Open Source Edition can be obtained from glassfish.org as a ZIP archive. Once expanded, this archive provides a fully functional application server. No further initial configuration is required, and we can start the default configuration straightaway:

```
$ cd glassfish
$ bin/asadmin start-domain
```

It is probably a good idea to put the bin/ directory under your global PATH environment variable so that the command-line tool asadmin becomes available from anywhere. We will use asadmin extensively to administer GlassFish. This is not the only way to do so, because GlassFish also provides Web and Representational State Transfer (REST) interfaces that are on par with the command set of asadmin.

The name of the default domain provided with the GlassFish distribution is domain1. The start-domain command should be followed by the name of the domain you want to start, but the domain name can be omitted when there is only one domain. Conversely, the stop-domain command is used to stop a previously started domain.

PRODUCTION READY

The capabilities of GlassFish do not stop after the development phase. It is also a production-class Java EE application server.

You can create your own domains using the create-domain command. You can even have several of them running on the same physical host as long as they have different network port settings.

On a single machine, such as a machine for developing applications, a domain can be seen as a GlassFish server configuration with its own parameters, including network ports, security policies, JDBC connection pools settings, deployed applications, and so on. Such a domain is called a *stand-alone domain*. However, in a clustered environment, such a domain is called a *Domain Administration Server* or *DAS*, and its role is to act as the central point for the management of a cluster. The next sections further explain this concept, but now let us get back to our running example.

LISTING 1 LISTING 2

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<!DOCTYPE html>
<html>
<head>
  <title>Hello World!</title>
</head>
<body>
<f:view>
<p>
  The current time is:
  <em>
    <h:outputText value="#{clock.now}"/>
  </em>
  <a href="index.jsp">(reload)</a>
</p>
</f:view>
</body>
</html>
```

See all listings as text

Deploying ClockEE to GlassFish is simple. From the source code folder, we need only build a WAR archive through Maven, and use asadmin to perform the deployment:

```
$ mvn package
...
$ asadmin deploy
target/clockee-1.0-SNAPSHOT.war
Application deployed with name
clockee-1.0-SNAPSHOT.
Command deploy executed
successfully.
```

You can now pick your favorite Web browser and point it to http://localhost:8080/clockee-1.0-SNAPSHOT/faces/index.jsp to see a page similar to that of **Figure 1**. You can also list the currently deployed applications by using the following command:

```
$ asadmin list-applications
clockee-1.0-SNAPSHOT  <web>
Command list-applications
executed successfully.
```

Now that we have our GlassFish

installation running properly, we can undeploy ClockEE before continuing with the clustering facilities:

```
$ asadmin undeploy
clockee-1.0-SNAPSHOT
Command undeploy executed
successfully.
$ asadmin list-applications
Nothing to list.
Command list-applications
executed successfully.
```

Our First Cluster

As mentioned earlier, a GlassFish DAS can be used to manage a cluster of GlassFish instances. The asadmin start-domain command actually starts a DAS.

A DAS is able to manage several local and remote instances of a GlassFish server, thus effectively providing a cluster. This makes it possible to manage a cluster from a single host by issuing commands to asadmin and having them be propagated automatically to perform application deployment, configuration, and resource management.

Let's get started by creating a cluster on our local machine. First off, ensure you have a domain and, hence, a DAS running. You can then use the create-cluster command, as shown in **Listing 3**, to create my-first-cluster.

While my-first-cluster has been successfully declared to the domain1 DAS, it is of little use because it does not have any GlassFish instances:

```
$ asadmin list-instances
Nothing to list.
Command list-instances
executed successfully.
```

Fortunately, creating instances is very easy. Let's create two local instances as part of my-first-cluster, that is, two instances running on the very same host as the one that we are using for the DAS. See **Listing 4**.

The output of this command is very instructive: each local instance has its own set of network ports. The first instance has its HTTP port on 28080, while the second one has it on 28081. There is, consequently, no network port conflict with the default assignments.

Of course, you could specify your own port numbers if you wanted to, but in most cases you will be just fine with the default ones. We can check that our instances are now part of my-first-cluster, yet not running, as shown in **Listing 5**.

Starting the cluster both makes it active and launches the instances. See **Listing 6**.

We can also check which nodes are parts of our cluster and which instances are attached to them, as shown in **Listing 7**.

GET SOME REST

A REST Web service interface is available for easily integrating GlassFish with third-party management systems. There is also a Web administration console that provides an intuitive graphical interface.

LISTING 3 / LISTING 4 / LISTING 5 / LISTING 6 / LISTING 7

```
$ asadmin list-domains
domain1 running
Command list-domains executed successfully.
$ asadmin create-cluster my-first-cluster
Command create-cluster executed successfully.
$ asadmin list-clusters
my-first-cluster not running
Command list-clusters executed successfully.
```

See all listings as text

By creating a local instance, GlassFish automatically created a local *config node*. (We will later explore the options available when creating instances for remote systems.)

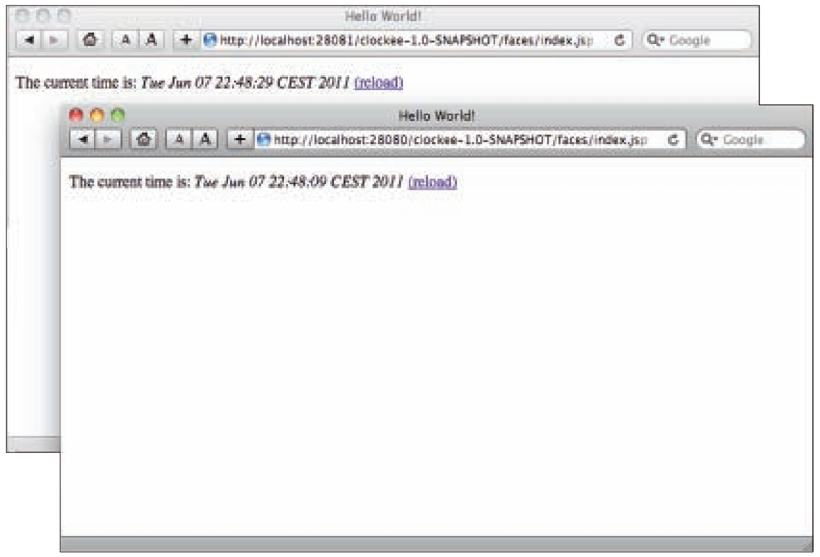
It is now time to deploy ClockEE to our cluster:

```
$ asadmin deploy --target
my-first-cluster
```

```
target/clockee-1.0-SNAPSHOT.war
Application deployed with name
clockee-1.0-SNAPSHOT.
Command deploy executed
successfully.
```

We can check that it is running, both by using asadmin and by pointing our Web browser to each instance HTTP connector, as depicted in **Figure 2**.

```
$ asadmin list-applications
my-first-cluster
clockee-1.0-SNAPSHOT  <web>
Command list-applications
executed successfully.
```



Hello World!
http://localhost:28081/clockee-1.0-SNAPSHOT/faces/index.jsp
The current time is: *Tue Jun 07 22:48:29 CEST 2011* (reload)

Hello World!
http://localhost:28080/clockee-1.0-SNAPSHOT/faces/index.jsp
The current time is: *Tue Jun 07 22:48:09 CEST 2011* (reload)

Figure 2

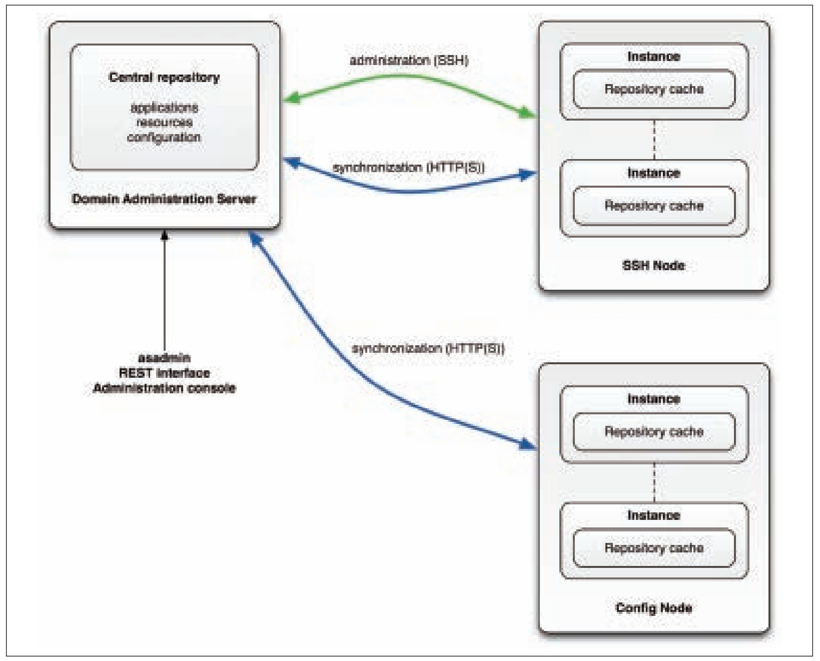


Figure 3

Creating a cluster and deploying a simple application with so little effort to a server that we just downloaded is already no small achievement.

You can, of course, have much more fine-grained control over the instances of a cluster. For example, you can start and stop them individually, as shown in **Listing 8**.

LISTING 8

```
$ asadmin stop-instance local-instance-1
The instance, local-instance-1, is stopped.
Command stop-instance executed successfully.
$ asadmin list-instances -l
NAME              HOST       PORT   PID    CLUSTER            STATE
local-instance-1  localhost  24848  -1     my-first-cluster   not running
local-instance-2  localhost  24849  16548  my-first-cluster   running
Command list-instances executed successfully.
$ asadmin start-instance local-instance-1
Waiting for local-instance-1 to start ...............
Successfully started the instance: local-instance-1
instance Location: /usr/local/Cellar/glassfish/3.1/libexec/glassfish/nodes/local-
host-domain1/local-instance-1
Log File: /usr/local/Cellar/glassfish/3.1/libexec/glassfish/nodes/localhost-do-
main1/local-instance-1/logs/server.log
Admin Port: 24848
Command start-local-instance executed successfully.
The instance, local-instance-1, was started on host localhost
Command start-instance executed successfully.
$ asadmin list-instances -l
NAME              HOST       PORT   PID    CLUSTER            STATE
local-instance-1  localhost  24848  16846  my-first-cluster   running
local-instance-2  localhost  24849  16548  my-first-cluster   running
Command list-instances executed successfully.
```

See all listings as text

Finally, we can undeploy the application from the cluster:

```
$ asadmin undeploy
--target=my-first-cluster
clockee-1.0-SNAPSHOT
Command undeploy
executed successfully.
```

GlassFish Clustering Terminology

Now that we have had an introduction, it is time to look at the terminology used for GlassFish clustering.

Figure 3 shows an overview of the DAS architecture. The DAS maintains a central repository containing deployed applications, declared resources, and configuration.

When we deployed ClockEE in the previous sections, we actually deployed it to this central repository by using asadmin. Similarly, we could have declared a resource such as a JDBC connection pool, and it would have been declared in the DAS central repository.

When used in a cluster, the DAS maintains a list of *nodes* taking part in a cluster configuration. Each node represents a server where *instances* are actual GlassFish servers.

When we created the two local instances earlier, GlassFish created a node on our behalf on the current machine. Every instance has a cache of the central repository. Synchronization between the DAS and the node instances is performed over HTTP or HTTPS (see the asadmin enable-secure-admin command). As an example, declaring a new resource in the DAS is propagated to the node instances when synchronization occurs.

Two distinct types of nodes exist:

- *Config nodes* need to be locally provisioned, configured, and administered. They are declared to a cluster from the DAS, but the DAS cannot perform any administrative commands. Administrators are required to log in to the machines running such nodes and perform the required tasks. Such nodes cannot be centrally administered from the DAS.
- *SSH nodes* can be completely managed from the DAS. The minimal requirement is to have an SSH server and a Java SE runtime environment on such nodes. Another benefit of SSH nodes is that communications between the DAS and the nodes is secure, both from an authentication and a confidentiality point of view.

NODE NOTE

SSH nodes can be **completely managed** from the DAS.

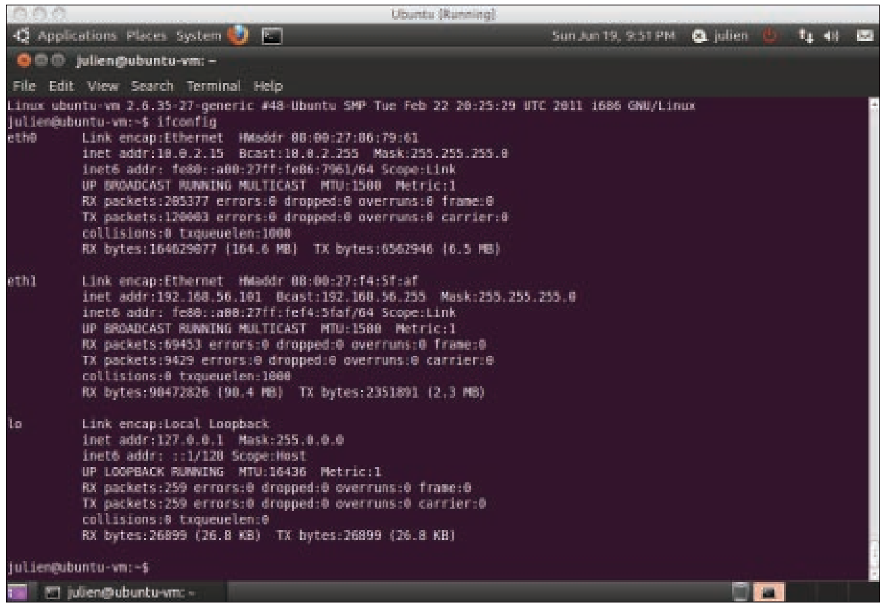
SSH nodes are preferable, whenever possible, because the complete management of a GlassFish cluster can be performed from the DAS host machine.

Given that SSH nodes arguably provide more-interesting features, we will focus solely on them in the remainder of this article. Adding Config nodes to a cluster requires preparing the node hosts by installing GlassFish, creating local Config nodes on each of the hosts, and declaring the nodes to the DAS. As the next section shows, these tasks can be fully automated in the case of SSH nodes.

Remotely Provisioning and Adding an SSH Node

In this section, we are going to see how to remotely provision and add a GlassFish instance to our cluster. It is assumed that you have a valid user account on a remote server that has, at the very least, a working Java SE 6 runtime environment and an OpenSSH server.

For the purpose of testing, you could do this on your local machine as well, if OpenSSH is running. A solid alternative that I experimented with while writing this article is to use a virtual machine. I used Oracle VM VirtualBox to virtualize a Linux Ubuntu Maverick operating system (see **Figure 4**).


Ubuntu [Running]
Applications Places System
Sun Jun 19, 9:51 PM
julien
julien@ubuntu-vm: ~
File Edit View Search Terminal Help

```
Linux ubuntu-vm 2.6.35-27-generic #48-Ubuntu SMP Tue Feb 22 20:25:29 UTC 2011 i686 GNU/Linux
julien@ubuntu-vm:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:86:79:61
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe86:7961/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:205377 errors:0 dropped:0 overruns:0 frame:0
          TX packets:120003 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:164629077 (164.6 MB)  TX bytes:6562946 (6.5 MB)

eth1      Link encap:Ethernet  HWaddr 08:00:27:f4:5f:af
          inet addr:192.168.56.101  Bcast:192.168.56.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fef4:5faf/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:69453 errors:0 dropped:0 overruns:0 frame:0
          TX packets:9429 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:90472026 (90.4 MB)  TX bytes:2351091 (2.3 MB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:259 errors:0 dropped:0 overruns:0 frame:0
          TX packets:259 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:26899 (26.8 KB)  TX bytes:26899 (26.8 KB)

julien@ubuntu-vm:~$
```

Figure 4

Given that operations happen over SSH, any action that requires connecting to the remote servers will require authentication. By default you will have to type your password every time, but you can drastically simplify this. Password-less SSH connections are possible by generating a public/private key pair locally and adding the public key fingerprint to the ~/.ssh/authorized_keys file on the remote servers.

Although you can do this manually, GlassFish can automate it, as shown in **Listing 9**.

LISTING 9 / LISTING 10 / LISTING 11 / LISTING 12

```
$ asadmin setup-ssh --sshuser
julien 192.168.56.101
Enter SSH password for julien@192.168.56.101>
Copied keyfile /Users/julien/.ssh/id_rsa.pub to julien@192.168.56.101
Successfully connected to julien@192.168.56.101 using keyfile
/Users/julien/.ssh/id_rsa
Command setup-ssh executed successfully.
```

See all listings as text

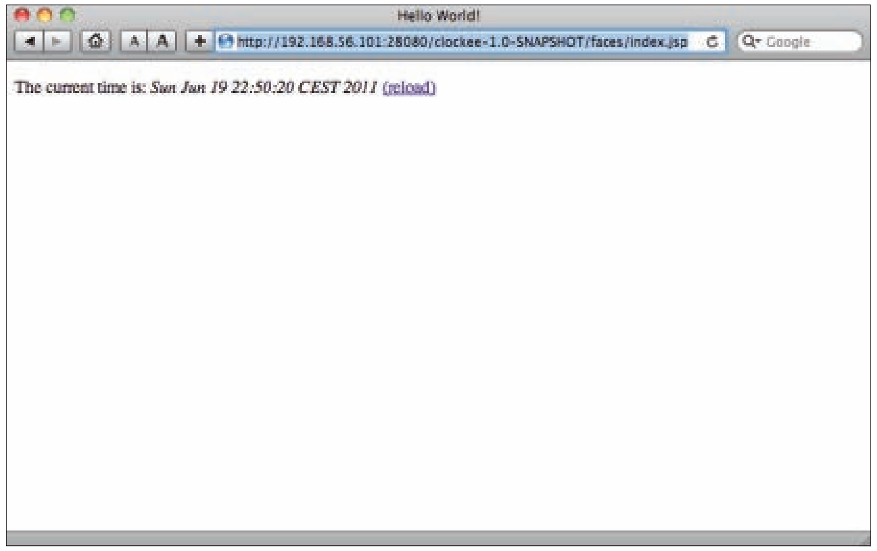
Let's assume that the remote server does not have GlassFish. Let's install it on the remote machine in /home/julien/glassfishv3 by performing a remote provisioning. See **Listing 10**.

A working GlassFish application has just been provisioned remotely from a local image that was copied over SSH. We can now create an SSH node on the remote server:

```
$ asadmin create-node-ssh
--nodehost 192.168.56.101
--installdir
/home/julien/glassfishv3 ubuntuvm
Command create-node-ssh executed
successfully.
```

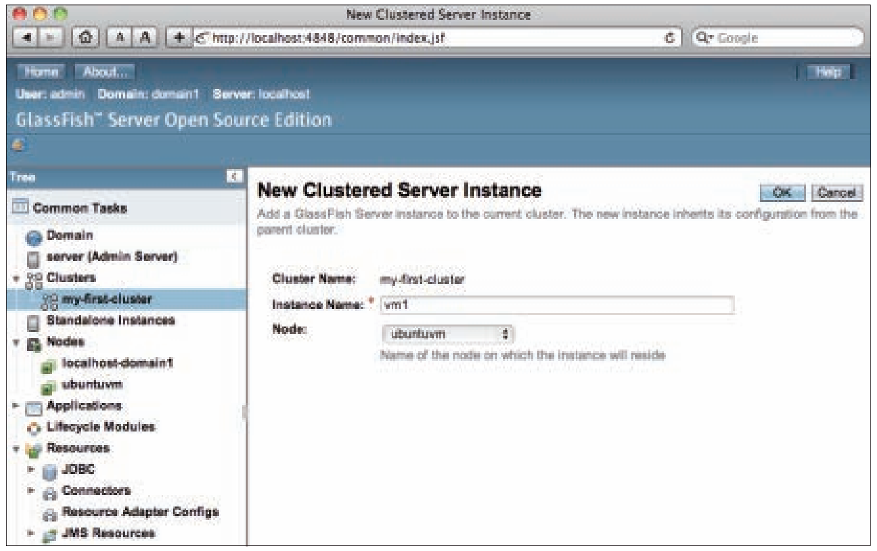
As shown in **Listing 11**, we can create an instance on this SSH node, name it vm1, and then add it to my-first-cluster.

By default, the newly created instance is not running, so we can start it. See **Listing 12**.


Hello World!
http://192.168.56.101:28080/clockee-1.0-SNAPSHOT/faces/index.jsp
Google

The current time is: *Sun Jun 19 22:50:20 CEST 2011* (reload)

Figure 5

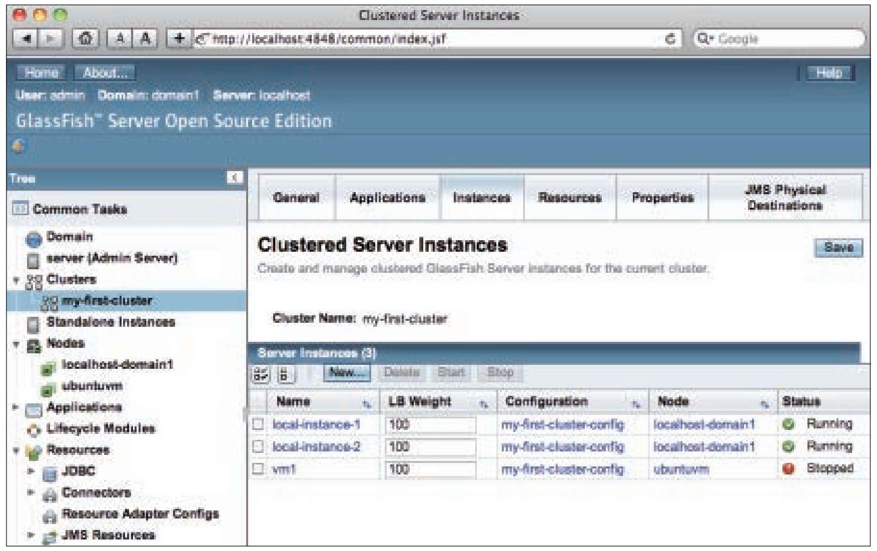


New Clustered Server Instance

Add a GlassFish Server instance to the current cluster. The new instance inherits its configuration from the parent cluster.

Cluster Name: my-first-cluster
Instance Name: * vm1
Node: ubuntuvm
Name of the node on which the instance will reside

Figure 6



Clustered Server Instances

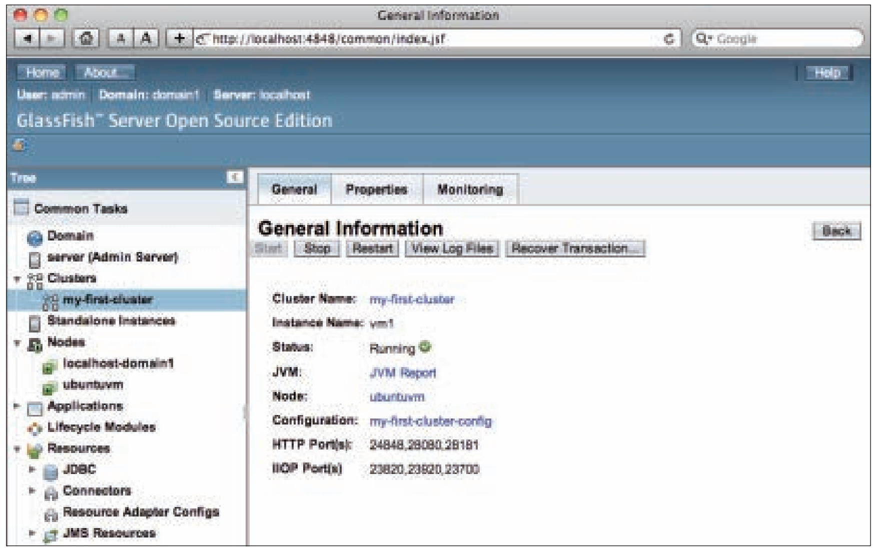
Create and manage clustered GlassFish Server instances for the current cluster.

Cluster Name: my-first-cluster

Server Instances (3)

Name	LB Weight	Configuration	Node	Status
local-instance-1	100	my-first-cluster-config	localhost-domain1	Running
local-instance-2	100	my-first-cluster-config	localhost-domain1	Running
vm1	100	my-first-cluster-config	ubuntuvm	Stopped

Figure 7



General Information

Cluster Name: my-first-cluster
Instance Name: vm1
Status: Running
JVM: JVM Report
Node: ubuntuvm
Configuration: my-first-cluster-config
HTTP Port(s): 24848,28080,28181
IIOP Port(s) 23820,23920,23700

Figure 8

We can check that ClockEE was transparently deployed to the vm1 instance when it was launched, provided that it had already been deployed to my-first-cluster (see **Figure 5**).

How About the Web Administration Console?

We did all administration work from the command-line using asadmin, but as was mentioned earlier, that is not the only option. A REST Web service interface is available for easily integrating GlassFish with third-party management systems. There is also a Web administration console that provides an intuitive graphical interface. Both are on par with the equivalent asadmin shell commands.

Figure 6, **Figure 7**, and **Figure 8** provide a few screenshots of using the Web administration console to perform a few tasks we did from the command-line before.

STAY TUNED

The next article shows how to enable high availability of stateful applications on the cluster.

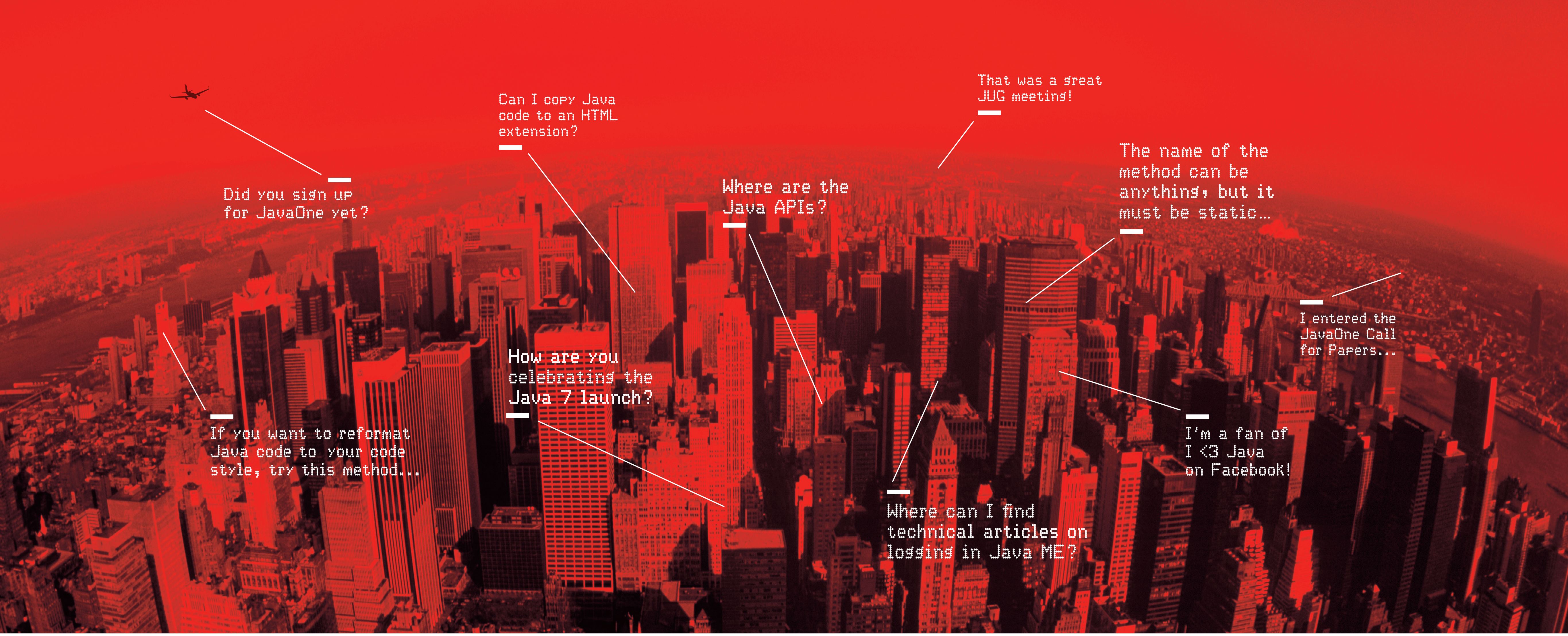
Conclusion

This article introduced the centralized cluster provisioning and management of GlassFish clusters. The ability to get a new host up and running remotely as long as it has an SSH server and a Java SE runtime is especially powerful. We deployed a simple stateless application to a cluster and played with common administration commands to check the state of our cluster and manipulate instances.

The next article in this two-part series shows how to enable high availability of stateful applications deployed to our cluster by providing transparent session failover. `</article>`

LEARN MORE

- GlassFish product documentation
- "Clustering in GlassFish Version 3.1"
- GlassFish videos on YouTube
- *Real World Java EE Night Hacks—Dissecting the Business Tier* by Adam Bien (press.adam-bien.com, 2011)
- *Beginning Java EE 6 with GlassFish 3* by Antonio Goncalves (Apress, 2010)
- *The Java EE 6 Tutorial: Basic Concepts* fourth edition by Eric Jendrock, Ian Evans, Devika Gollapudi, Kim Haase, and Chinmayee Srivathsa (Prentice Hall, 2010)


Did you sign up for JavaOne yet?
Can I copy Java code to an HTML extension?
Where are the Java APIs?
That was a great JUG meeting!
The name of the method can be anything, but it must be static...
I entered the JavaOne Call for Papers...
If you want to reformat Java code to your code style, try this method...
How are you celebrating the Java 7 launch?
Where can I find technical articles on logging in Java ME?
I'm a fan of I <3 Java on Facebook!

Oracle Technology Network: Your Java Nation.

Come to the best place to collaborate with other professionals on everything Java.

Oracle Technology Network is the world's largest community of developers, administrators, and architects using Java and other industry-standard technologies with Oracle products. Sign up for a free membership and you'll have access to:

- Discussion forums and hands-on labs
- Free downloadable software and sample code
- Product documentation
- Member-contributed content

Take advantage of our global network of knowledge.

JOIN TODAY ▸ Go to: oracle.com/technetwork/java

VINICIUS SENGER AND
YARA SENGER
BIO

Wake Up and Smell the Coffee

The creators of jHome, a 2011 Duke's Choice Award winner, demonstrate why Java EE 6 development is so easy.

jHome is a complete home-automation open source API based on GlassFish Server Open Source Edition and Java EE 6, which enables developers to control anything in their homes.

In this article, we describe the jHome project in detail, including the technical context, code, and hardware details. Usage examples show the power and ease of use of Java EE.

Like most Java programmers, we are crazy for coffee. We always wanted to schedule our coffee maker to turn on at a certain time, so we could wake up to the smell of coffee. So we set out to develop a way to do this. After all, nothing is more fun than using the Java EE 6 Timer Service to schedule the "real objects" of your life.

People who always lose or forget their keys, for instance, might want to control their front door through a Website or a mobile application. And why not open your door by sending a message to a Twitter profile that is associated with your jHome application? Others might want to control their balcony lights or swimming pool lights from their mobile devices. The possibilities for simplifying the way you control the objects in your life are unlimited.

jHome Overview

jHome is a set of interfaces and components based on Java EE 6 for implementing home-automation solutions. Using Enterprise JavaBeans (EJB), the Timer Service, Java API for RESTful Web Services (JAX-RS), Contexts and Dependency Injection (CDI), and Web components, jHome lets you automate your house, build a Web or mobile app, and control things over the internet.

Figure 1 shows an overview of the jHome architecture.

With the simple code shown in **Listing 1**, you can inject an object that represents a lamp in your house and turn it on or off using a Servlet.

Using EJB beans to control a lamp may sound strange, but the Singleton, Transactions, and Injections resources are a very useful way to deal with hardware communication, concurrency, and queuing requests.

jHome Devices

jHome is based on open source hardware and does not require proprietary or expensive solutions. The jHome team created the reference implementation hardware based on Arduino, an open source electronics prototyping platform. The cost to control two lamps or wall sockets is less than US$100 using Arduino and USB cables.

Each jHome device can communicate with the jHome application deployed on the Java EE container using different protocols and different wired and wireless components:

- USB cable (the cheapest solution)
- Bluetooth (good for wireless prototypes)
- ZigBee/xBee (the best wireless solution for real implementations)

Relay boards are very useful because they let you control wall sockets, lamps, and any other electronic device. jHome can also read values from sensors such as temperature, light, distance, and gas.

The standard firmware for jHome devices can control relays, dimmers, RGB LEDs, motors, and robots, and code for various sensors is available at the Tools Cloud

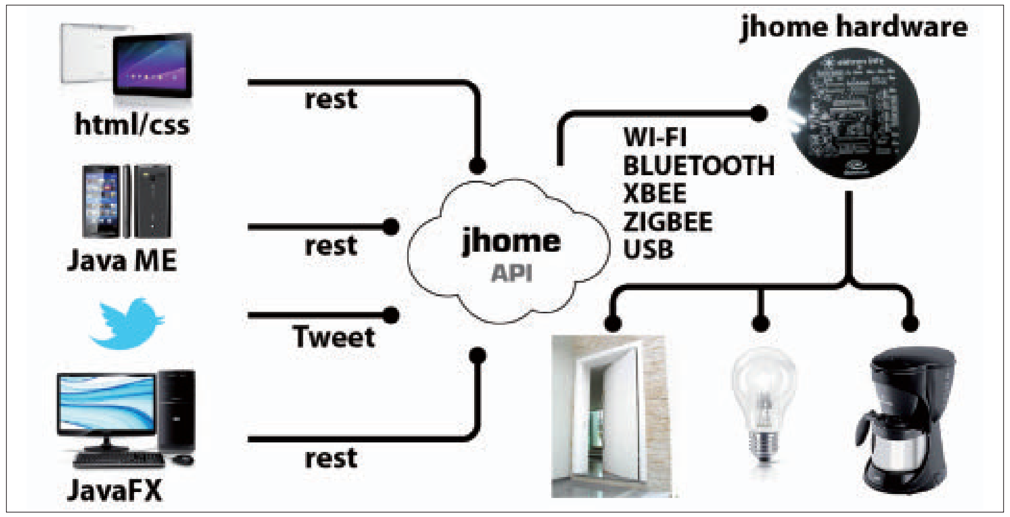


Figure 1

GIT repository. Use username jhome and password: jhome. Let's clone!

Another interesting implementation we made was to integrate jHome with a heart rate transmitter and program jHome to do a certain task (such as open a Website or send a message) when the user achieves a predefined maximum or minimum heart rate. In the Java EE world this would be a Heart-Driven Bean—something that brings the fun back to development.

Understanding Arduino and Open Source Hardware

Arduino is an open source electronics prototyping platform based on flexible, easy-to-use hardware (a printed circuit board) and software. There are several different implementations varying in size, price, and memory capacity.

The programming language used to program the board is a C-like language, and there is a Java IDE that helps you to create programs and send them to the board. Although the language running on the Arduino board is not Java, the syntax is easy for experienced Java programmers to learn and understand.

Java EE 6 and jHome

Java EE has really evolved over the last 10 years. It's still powerful but is now much easier to use, affordable, and lightweight—in fact, our cell phones can now run a Java EE container, such as GlassFish Server Open Source Edition.

Controlling your house with a Java EE application server is now a reality. All Java EE components are useful for automating control of your house. Here's how the jHome team used some of the components in the jHome implementation.

EJB: core components. EJB beans are used for all jHome core components. The EJB resources used in this application are the following:

- Singleton: Used to represent devices and boards that cannot have more than one instance
- Transaction: Used to deal with hardware concurrency
- Web services: Very useful to expose functionality to any kind of client application
- Timer Service: Used to schedule wall sockets, update sensor values, check for Twitter messages, and so on

Timer Service: scheduling the coffee maker. jHome uses the Timer Service API to allow you to schedule services such as turning lights on and off and scheduling the coffee maker.

And because coffee is a very critical resource for Java developers, it's important to have high availability. The Timer Service API delivers this: if you restart your application server, you will not need to reschedule the coffee maker.

Servlet: controlling lights via the internet. There are a couple of Servlets for controlling lights. There are LED strips that can control color using RGB. You can control the color of your swimming pool lights, aquarium lights, or any other light.

jHome Architecture

jHome is modular and uses Maven. Let's look at its modules.

LISTING 1 LISTING 2

```
@WebServlet(name = "Light", urlPatterns = {"/Light"})
public class Light extends HttpServlet {
  private static boolean on;

  @EJB
  Relay light;

  @Override
  protected void doGet(HttpServletRequest request,
                       HttpServletResponse response)
            throws ServletException, IOException {
      if (on) {
       light.turnOff("lamp");
       on = false;
      } else {
       light.turnOn("lamp");
       on = true;
      }
  }
}
```

See all listings as text

jHome architecture. jHome has three main modules:

- jHome Interface : All jHome interfaces can be found in this module.
- jHome Core: This module has all the interface's implementation and is responsible for discovering the jHome devices available in the house using an open protocol, reading their sensors, and also helping to expose the jHome device functionality to other Java components.
- jHome Web: A Web application that uses HTML5 and jQuery to control the default functionalities from jHome: turn a lamp on and off, control the RGB color LEDs, schedule a coffee maker, and read basic sensors.

You must deploy the jHome Interface and Core modules to run the basic jHome services. The jHome Web module is optional, and you can create or use any other client applications such as Java ME, JavaFX, or Ginga-J to run on your TV.

Sensor API. The Sensor API provides information about the physical environment, and it is implemented using the Timer Service.

With the jHome Sensor API you can feed your Java application or Website with information from different sensors that are plugged into your jHome devices. This capability provides many new possibilities for designing your

Website or Java application, for instance:

- Change the background color of your Website based on information from a temperature sensor.
- Make a component turn on a light based on information from a light sensor.
- Give the dimensions of your house to a 3-D CAD system using a distance sensor.

Social network APIs. The jHome Twitter API lets you control your home by sending tweets mentioning a specific Twitter profile and using a Twitter hash. For example, we can control our office using the Twitter commands shown in **Listing 2**. The bold words in the code must be included in your tweet. They are part of the syntax, used to identify the action. The other words are optional in the tweet.

You can create your own extension and create a Facebook connector or implement any other interesting idea.

Home Automation as a Service

It is impossible not to think about the relation between any new project and the cloud nowadays. Some people might think that maintaining a Java EE container/server in their houses to turn lights on and off, schedule a coffee maker, and open doors/gates can be a bit complex. They might want to try it on the cloud to further enhance their experience with the project.

jHome fits very well on the cloud and can be provided as software as a service (SaaS).

Our team developed a jHome Proxy Device, which is a very simple board with an Ethernet connection that you can plug into your internet router. It will receive jHome instructions from the cloud.

Even if the cloud server is unavailable, you can access this device via HTTP. It's a prototype, but it's working, and there seems to be a good market for it.

Conclusion

jHome demonstrates that Java EE 6 is mature, simple, easy, and inexpensive to maintain. Additionally, with the advent of open source hardware, you can create your own solutions and automations.

The jHome team is proud to have won a 2011 Duke's Choice Award and received community recognition for the jHome project.

Remember, there is life beyond database applications. </article>

LEARN MORE

- jHome
- Arduino

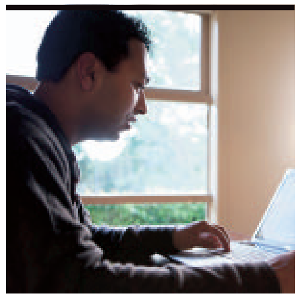
3 Billion Devices Run Java

Computers, Printers, Routers, BlackBerry Smartphones, Cell Phones, Kindle E-Readers, Parking Meters, Vehicle Diagnostic Systems, On-Board Computer Systems, Smart Grid Meters, Lottery Systems, Airplane Systems, ATMs, Government IDs, Public Transportation Passes, Credit Cards, VoIP Phones, Livescribe Smartpens, MRIs, CT Scanners, Robots, Home Security Systems, TVs, Cable Boxes, PlayStation Consoles, Blu-ray Disc Players…

Java™ | #1 Development Platform

ORACLE®

oracle.com/goto/java
or call 1.800.ORACLE.1



VIKRAM GOYAL

Developing Proximity Awareness with the Location API

Using the Location API, you can access location data and incorporate that data into a game.

This article is the second article in a two-part series. Part 1 described how to set up the GameCanvas and deal with the Mobile Sensor API, JSR-256. We still need to add proximity (location) awareness aspects to our location-based game. In this article, we add the location-based code.

For the basics of the Location API, JSR-179, and the Mobile Sensor API, see my previous tutorials, which are listed in the "Learn More" section of this article. In addition, the demo apps provided with the Java ME SDK are very relevant and provide good insight into the working of these APIs.

Note: The source code for the game I develop in this two-part series can be downloaded here.

WHAT'S NEARBY?
Determine what is near you using Bluetooth technology, Wi-Fi networks, cell phone tower triangulation, and, of course, GPS.

Being Location-Aware

Determining what is near you can be done in several ways: Bluetooth technology, Wi-Fi networks, cell phone tower triangulation, and, of course, GPS. What might work in one situation might not work best in others. A combination of these technologies is perhaps the best bet for figuring out the location of your device.

In this second article, I concentrate on using the Location API, because it provides the easiest way of finding people and items that are near you. Furthermore, you can broadcast your own location easily enough, so others can find you as well.

Using the Location API

Listing 1 shows the initialization code for the Location API criteria in the constructor of the MIDlet.

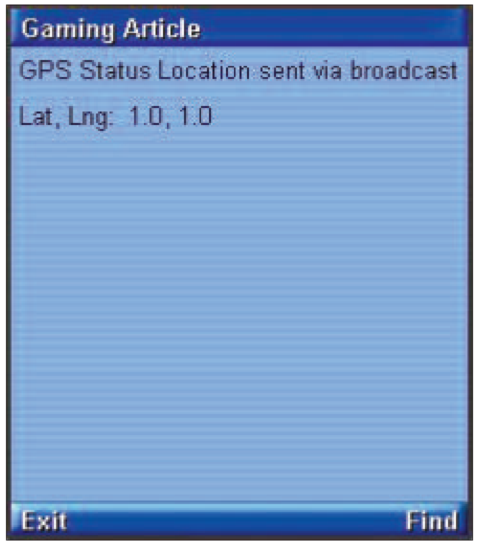
We define a very basic set of criteria for the Location API. We have no special requirements for the horizontal or vertical accuracies or the response time, but we don't want any costs incurred in acquiring the location data. This should suffice in even the most basic GPS devices.

Next, we acquire the LocationProvider using these criteria. Of course, if the provider cannot be found, we cannot proceed further and a message is displayed to the gamer.

Our MIDlet should implement the LocationListener interface, because if the provider is found, the MIDlet can listen to the events that the provider generates. The most important event is, of course, updates on current coordinates. The locationUpdated method implementation handles this in our code, as shown in **Listing 2**.

Once we have a valid location, we display that information using the getQualifiedCoordinates method. The location is displayed to the user, and it is also broadcast to the server so that it can be registered, as shown in **Figure 1**.

The next step is to find players near your own location with whom you can play the game, as shown in **Listing 3**. The getPlayersFromServer placeholder mock method returns a list of



Gaming Article
GPS Status Location sent via broadcast
Lat, Lng: 1.0, 1.0

Figure 1

PHOTOGRAPH BY JONATHAN WOOD/ GETTY IMAGES

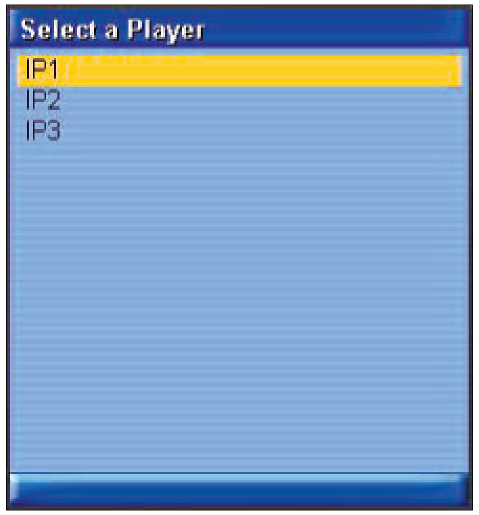

Select a Player
IP1
IP2
IP3

Figure 2

players. The findPlayers command is handled by the commandAction method.

For the best results, make sure any code that might take a long time to return, such as querying the server for a list of players, is handled in a separate thread so it doesn't block the main program flow. The end user should be able to cancel the query from the server if required, and the program should handle that implicitly. **Figure 2** displays a list of players.

Handling the OK button for the selection of player launches the game, as shown in **Listing 4**.

The initiateGame mock notifies the server and then launches the game using the launchGame method, as shown in **Listing 5**.

For trivial applications, this should suffice, but for production applications, there are other aspects that would require attention as well, such as better error handling, increased sensitivity of detection, valid response from the server, and so on.

LISTING 1 / LISTING 2 / LISTING 3 / LISTING 4 / LISTING 5

```
try {

  // create very basic criteria for the location provider
  Criteria criteria = new Criteria();
  criteria.setHorizontalAccuracy(Criteria.NO_REQUIREMENT);
  criteria.setVerticalAccuracy(Criteria.NO_REQUIREMENT);
  criteria.setPreferredResponseTime(Criteria.NO_REQUIREMENT);
  criteria.setCostAllowed(false);

  // get the provider based on the criteria
  LocationProvider provider = LocationProvider.getInstance(criteria);

  // provider found
  if (provider != null) {

   // set this MIDlet to be the listener for location changes
   provider.setLocationListener(this, -1, 0, 0);

  } else {
   display.setCurrent(
    new Alert("Error!",
    "Phone does not support Location Provider with the given criteria",
    null, AlertType.ERROR));
  }

} catch (Exception ex) {
  handleError(ex);
}
```

See all listings as text

Conclusion

In this second article, we discussed how to incorporate the Location API into the gaming code that we developed in the previous article. We also saw that our core MIDlet needs to implement the LocationListener interface and handle the LocationUpdated method (at the very least). In addition, we discussed how to form very basic criteria for accessing location data and we saw how to incorporate the data we get into our game. </article>

LEARN MORE

- Tutorial: "Using the Location API for Favorite Spots"
- Tutorial: "Working with the Mobile Sensor API"

Using Maven to Compare JVM Scripting Languages

Learn how to use Maven to quickly try your hand at Clojure, Groovy, JavaScript, JRuby, and Jython.

CASIMIR SATERNOS

BIO

Matt Mullenweg, a highly successful entrepreneur and founding developer of WordPress, recently remarked that "Scripting is the new literacy, and the ability to learn and execute on your ideas . . . is going to be invaluable."

Scripting languages can decrease time to market, improve efficiency, boost performance, and automate previously error-prone manual workflows. A new programming language provides a new framework to conceptualize ideas and might offer otherwise inaccessible solutions. In particular, less-structured and ad hoc projects can be implemented more effectively by choosing the right scripting language. Specific APIs or libraries that were implemented for a given language might not be available for other languages.

This article presents a project, based in Apache Maven, that can be set up in minutes and allows developers to run and compare scripts written in five dynamic scripting languages: Clojure, Groovy, JavaScript, JRuby, and Jython.

Note: The project was developed and tested on Microsoft Windows and Mac OS using the following software versions:

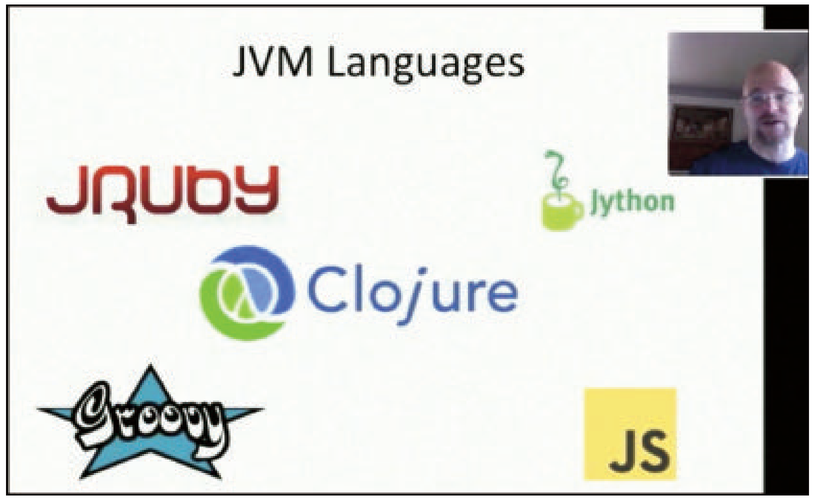
- Java 1.6.0_14-b08, 1.6.0_24-b07-334, 1.5.0_19-b02-304
- Apache Maven 3.0.3 (r1075438)
- Apache Maven 3.0.2 (r1056850)
- Apache Maven 2.2.1 (r801777)

Language Differences

Programming languages, which are designed with different goals in mind, vary in syntax, target platforms, performance, and features. Choosing a language with a desirable syntax is important. Some languages emphasize expressiveness and clarity, while others focus on conciseness. Syntax concerns might not even be purely technical. Yukihiro Matsumoto has said that he created Ruby with the intention of giving programmers joy.

Initially, a language might target a specific platform. JavaScript gained popularity in Web browsers. Python is used as the scripting language for open source projects such as Blender. The Java-based Python implementation known as Jython is used in commercial software such as Oracle Data Integrator. Certain cloud computing and hosting services provide support for specific languages. Heroku supports Clojure, JavaScript, and Ruby, while the Google App Engine utilizes Go, Java, and Python. So your choice of language might well be dictated by the arena of your development efforts.

Using different languages offers new conceptualizations of problems. Learning Clojure increases one's appreciation for elegance of design and the use of recursion. Developers will never think of whitespace the



Learning a new language gives you the power to create better software, says Casimir Saternos.

same way after coding in Python, where whitespace is syntactically significant.

The stereotype of the lone-wolf programmer locked away in a room has faded in recent years. Team software development is the norm, and pair programming is an accepted Agile practice. Communities with different strengths grow up around a language. With such considerations in mind, developers might want to set up an environment where they can test-drive several new languages to quickly compare them. Java—specifically the Java Virtual Machine (JVM)—provides an excellent platform for such an exercise.

Java and the JVM

The Java language is not—and was never intended to be—the perfect programming language for every situation. The JVM is software constructed to execute bytecode compiled from Java. Each JVM implementation is designed to run on a specific platform. The original vision was to allow Java to be compiled once and run on a variety of platforms. But the JVM has also been leveraged in recent years to run bytecode created by other programming languages.

The JVM has accommodated other languages for some time, and support for such languages has been improved with each recent Java release. JSR-223 defined a scripting interface that was completed in Java SE 6. In Java SE 7, JSR-292 provides additional benefits including improved performance and additional support for dynamically typed languages. The JVM is mature, widely deployed, and heavily optimized, and it includes excellent tools and allows access to a large variety of libraries.

RUN AND COMPARE

The Apache Maven project presented here can be set up in minutes and allows developers to run and compare scripts written in five dynamic scripting languages: Clojure, Groovy, JavaScript, JRuby, and Jython.

Start Your Engines

Trying out Clojure, Groovy, JavaScript, JRuby, and Jython requires an initial setup in order to find, download, install, and configure each implementation. The project described here reduces the time and effort required by using Maven to locate and download resources from various public code repositories. Maven is also used to build the project so that a single executable JAR is created, which contains a class that can be used to run scripts written in several different languages.

Setup

To run the project, do the following:

1. Validate your Java installation by running the following commands:

```
javac -fullversion
java -fullversion
```

The output from each of these commands should indicate a Java SDK version of 1.6. If it does not, download and install a Java SDK (the Microsoft Windows version is jdk1.6.0_14).

2. Download and install Maven, and make sure it is included in your PATH environment variable. Then run the following command:

```
mvn -version
```

The output of this command should indicate that a recent version of Maven is installed (for example, version 3.0.3).

3. Download the project itself, which consists of a small amount of Java code, a few scripts, and a pom.xml file. Extract the code, and navigate to the directory that contains pom.xml.
4. Build the code, which will download JAR files from various archives and assemble the application:

```
mvn clean install
```

All downloaded JAR files are installed in your local Maven repository (by default, in your home directory under .m2/repository).

5. Run the application from within Maven (this works regardless of the platform in use). To see the list of available scripting languages, run the following command:

```
mvn -q exec:java
```

Note: You can also run the file by using the Java interpreter and referencing the JAR itself. The jvms .bat script contains such a call for a Microsoft Windows environment, and the jvms.sh script contains such a call for a Mac OS environment. The remainder of this article uses "mvn" to avoid operating-system-specific details.

6. To compare various languages, pass at the command line the scripts that are to be evaluated.

One or more files can be passed as arguments, and the files can use the same or different scripting languages. The scripts used in this article reside in src\main\resources\scripts. The customary "Hello World" example can be run as follows:

```
mvn -q exec:java -Dexec.args=
" hello.clj hello.js hello.groovy
hello.rb hello.py"
```

Additional Scripts

In addition to the "hello" scripts, several other scripts (described in a bit more detail below) are provided that can get you started with your own experiments.

You can run these scripts and modify them to compare and contrast Clojure, Groovy, JavaScript, Python, and Ruby. In some cases, the code is nearly identical, but in other cases, it is radically different. When you run a set of scripts, you will also notice that the output shows a time in nanoseconds for each script. This allows you to create scripts in several languages and immediately compare their performance.

Implementation Details

Maven. The convenience of building a sophisticated project by running a simple command is almost magical when first encountered. Maven enables developers to build a sophisticated project with a simple command through a build tool analogous to Apache Ant that offers a way to create standardized builds without significant knowledge of the underlying details. All the project information is found in the pom.xml file (also known as the Project Object Model or POM).

A POM and a set of plug-ins are all that are used by Maven to construct the project provided with this article. The repositories used in the project are listed toward the end of the POM. This enables Maven to obtain various JAR files, in this case, scripting language implementations and engines.

The dependencies section lists the specific JAR files that are included in the project. Some languages require two JAR files, while others require only one. This is because some languages have JSR-223 support built into the language itself, while others require a separate engine that provides JSR-223 compatibility.

BRIGHT FUTURE

The **JVM** is home to some of the top implementations of new languages, and it provides an **excellent infrastructure** for running them. With scripting language support improving in each new Java release, we can expect greater improvements and enhancements in the future.

Maven plug-ins. Maven uses a plug-in to perform a unit of work. A number of plug-ins are available that can be used to build a JAR file, including the following:

- maven-jar-plugin
- maven-assembly-plugin
- onejar-maven-plugin
- maven-shade-plugin

The first two plug-ins are standard and sufficient for many applications. The onejar-maven-plugin provides additional features to combine code and JAR files into a single JAR.

The project provided with this article faced the challenge that various language implementations use different versions of APIs that included different versions of classes with the exact same name. Fortunately, maven-shade-plugin can be used to automatically rename classes to avoid conflicts simply by specifying a transformer implementation class.

Java class. The Java application itself consists of under a hundred lines of Java code used to invoke the intended script engine based upon a file's extension.

If no arguments are passed to the application, it displays version information and available scripting language implementations. Otherwise, it reads the filenames passed to it as arguments and attempts to locate each file. (A path can be specified. The current working directory and src/main/resources/scripts are searched if no path is specified). If a file is found, the application loads a scripting engine associated with the file extension, and the script is run. Output generated by the script and related diagnostics are displayed.

A single test class, JvmsTest, is used during the build to validate that scripting engines can be identified for the filenames in question.

Included scripts. For each type of script listed below, a file with an associated extension exists:

- Clojure: .clj
- Groovy: .groovy
- JavaScript: .js
- Jython: .py
- Ruby: .rb

A number of scripts are included.

- hello: A customary "Hello World" program. This is included to provide a simple example of how to write to standard output.
- loop: A loop that outputs integers 0 through 9. This demonstrates procedural techniques of repetition and iteration. You can also use looping tests to evaluate performance.

The other two types of scripts use frequently implemented algorithms. They are particularly interesting in that they can be implemented in various ways that highlight differences between languages (using iteration, recursion, and various optimizations). See Rosetta Code, for example, for such comparisons. The included scripts are relatively equivalent to allow for valid baseline comparisons.

- fact: Outputs factorials (the product of positive numbers less than or equal to *n*) less than 100,000.
- fib: Outputs the numbers in the Fibonacci sequence under 100 (each number in the sequence is the sum of the previous two).

Conclusion

There are many advantages to understanding alternative languages for the JVM. The project presented in this article demonstrates how easily new languages can be incorporated into a project using Apache Maven.

The JVM is home to some of the top implementations of new languages, and it provides an excellent infrastructure for running them. With scripting language support improving in each new Java release, we can expect greater improvements and enhancements in the future. The best-prepared professionals will have a clear understanding of the resources available and how they can be best leveraged to address the technical needs driving business today. </article>

LEARN MORE

- Download Maven
- JSR-223, "Scripting for the Java Platform"
- JSR-292, "Supporting Dynamically Typed Languages on the Java Platform"

//fix this /

In the last issue, Arun Gupta gave readers a piece of code and asked if the code is guaranteed to work in a multithreaded environment.

The correct answer is #2: No. Servlets are not re-entrant, and EntityManager is not threadsafe. Instead inject as

```
@PersistenceUnit EntityManagerFactory em;
```

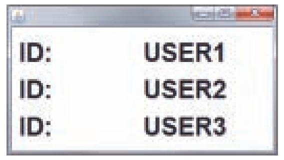
and then get EntityManager within the doGet() method.

This issue's challenge comes from Marek Piechut, a technical leader at Transition Technologies S.A. in Łódź, Poland, who is currently working on a heavy Swing client for a large enterprise application.

1 THE PROBLEM

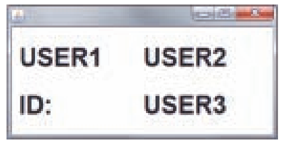
Java Swing is the main Java framework for heavy client implementations.

Let's create a Swing window that displays a simple table. What we want displayed is



ID:	USER1
ID:	USER2
ID:	USER3

But what we get is



USER1	USER2
ID:	USER3

2 THE CODE

Consider the following code when preparing a window:

```
public class MyWindow extends JFrame {
  public MyWindow() {
     GridLayout layout = new GridLayout(0, 2, 4, 4);
     setLayout(layout);
      JLabel idLabel = new JLabel("ID:");

      for (String d : new String[]{"USER1", "USER2", "USER3"}) {
          add(idLabel);
          add(new JLabel(d));
      }
  }
}
```

3 WHAT'S THE FIX?

1) The first value for GridLayout is invalid and should be replaced with the actual number of rows.

2) You can't add elements directly to JFrame; you should first pack all labels into JPanel and add it to JFrame.

3) You should replace a single shared instance of idLabel with a new JLabel instance for each row.

4) You need to call layout.layoutContainer(this) in the constructor end to lay out the window properly.

GOT THE ANSWER?

Look for the answer in the next issue. Or submit your own code challenge!

ART BY I-HUA CHEN

EDITORIAL
Editor in Chief
Justin Kestelyn
Senior Managing Editor
Caroline Kvitka
Community Editors
Cassandra Clark, Sonya Barry, Yolande Poirier
Java in Action Editor
Michelle Kovac
Technology Editors
Janice Heiss, Tori Wieldt
Contributing Writer
Kevin Farnham
Contributing Editors
Blair Campbell, Claire Breen, Karen Perkins

DESIGN
Senior Creative Director
Francisco G Delgadillo
Design Director
Richard Merchán
Contributing Designers
Jaime Ferrand, Paulina McFarland
Production Designer
Sheila Brennan

PUBLISHING
Publisher
Jeff Spicer
Production Director and Associate Publisher
Jennifer Hamilton +1.650.506.3794
Senior Manager, Audience Development and Operations
Karin Kinnear +1.650.506.1985

ADVERTISING SALES
Associate Publisher
Kyle Walkenhorst +1.323.340.8585
Northwest and Central U.S.
Tom Cometa +1.510.339.2403
Southwest U.S. and LAD
Shaun Mehr +1.949.923.1660
Northeast U.S. and EMEA/APAC
Mark Makinney +1.805.709.4745
Recruitment Advertising
Tim Matteson +1 310-836-4064
Mailing-List Rentals
Contact your sales representative.

RESOURCES
Oracle Products
+1.800.367.8674 (U.S./Canada)
Oracle Services
+1.888.283.0591 (U.S.)
Oracle Press Books
oraclepressbooks.com

ARTICLE SUBMISSION
If you are interested in submitting an article, please e-mail the editors.

SUBSCRIPTION INFORMATION
Subscriptions are complimentary for qualified individuals who complete the subscription form.

MAGAZINE CUSTOMER SERVICE
java@halldata.com **Phone** +1.847.763.9635

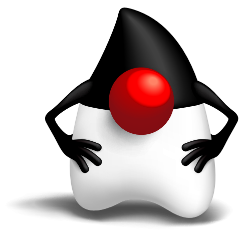
PRIVACY
Oracle Publishing allows sharing of its mailing list with selected third parties. If you prefer that your mailing address or e-mail address not be included in this program, contact Customer Service.

Java Magazine is published bimonthly with a free subscription price by Oracle, 500 Oracle Parkway, MS OPL-3C, Redwood City, CA 94065-1600.

Digital Publishing by Texterity

Learn Java 7 From the Source

— Oracle University —



- ✓ New Java SE 7 Training
- ✓ Engineering-Developed Courseware
- ✓ Taught by Oracle Experts
- ✓ 100% Student Satisfaction Program

Click for Details, Dates, and to Register

ORACLE®