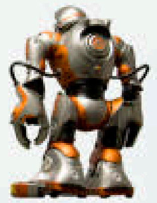
Java™ magazine

By and for the Java community

//MARCH/APRIL 2013 /

Java Is Community



ORACLE

//table of contents /

CORRECTION

In the September/October 2012 issue, we stated that jDays conference co-organizer Hamid Samadi was the founder of the Javaforum Gothenburg user group. This user group was actually founded by Rikard Thulin and Tomas Trolltoft. We regret the error.

COVER ART BY I-HUA CHEN

EDITORIAL

Editor in Chief
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
Claire Breen, Blair Campbell, Karen Perkins

DESIGN

Senior Creative Director
Francisco G Delgadillo

Senior Design Director
Suemi Lam

Design Director
Richard Merchán

Contributing Designers
Jaime Ferrand, Nicholas Pavkovic

Production Designers
Sheila Brennan, Kathy Cygnarowicz

PUBLISHING

Vice President
Jeff Spicer

Publisher
Jennifer Hamilton +1.650.506.3794

Audience Development and Operations Director
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

Advertising Sales Assistant
Cindy Elhaj +1.626.396.9400 x 201

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

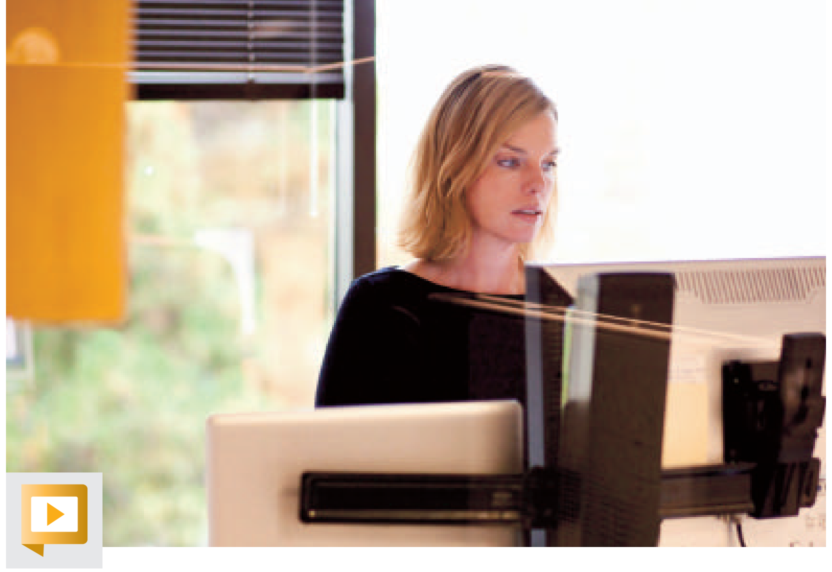
Learn Java On Demand

— Oracle University —

- ✓ Complete Classroom Content
- ✓ Highest Rated Instructors
- ✓ Immediate Online Access
- ✓ Search, Pause & Rewind Anytime

Preview Now

ORACLE®



A thriving community is vital to Java technology and the Java language. At *Java Magazine*, our job is to help you, our readers, become better Java developers and stronger community members. So in this community-focused issue, we give you a hands-on guide to Java citizenship—from taking the first steps to participation to taking on leadership positions and launching events.

In planning this issue, we took our tagline, "by and for the Java community," to heart. Who better to write content about building strong community than Java community members themselves? First, the editorial team named our first-ever guest editor: Agnès Crepet, Lyon (France) JUG leader and Duchess leader. We were inspired by her entrepreneurial spirit, sense of adventure, and passion for the Java community. Agnès signed on enthusiastically and has been a great resource to us in putting all of the pieces together for our "Java Is Community" cover story—from suggesting topics, to writing about reinvigorating your JUG, to helping us find photos. A big thank you to Agnès! Learn more about her in Kevin Farnham's Q&A.

Many other community leaders came together to make this issue a success as well. Thanks to Gail Anderson, Max Bonbhel, Tasha Carl, Badr El Houari, Bert Ertman, Ben Evans, Trisha Gee, Ahmed Hashim, Nety Herawaty, Michael Hüttermann, Hildeberto Mendonça, Yara Senger, Bruno Souza, Martijn Verburg, Richard Warburton, and Mila Yuliani. Plus a shout out to Oracle's Sharat Chander, Stephen Chin, and Nichole Scott for their participation. Don't miss Sharat in my editor's video.

We hope this issue inspires you to do great things with Java. The Java community keeps us inspired to make this magazine better with every issue.

Caroline Kvitka, 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

MAKE THE FUTURE JAVA

Java™

FIND YOUR JUG HERE

My local and global JUGs are great places to network both for knowledge and work. My global JUG introduces me to Java developers all over the world.

Régina ten Bruggencate
JDuchess

LEARN MORE

ORACLE®

Globalize Your Business

Melissa Data can help you globalize your applications as you expand operations to other countries or reach new customers in emerging markets. As a world leading data quality vendor, we offer solutions to verify, correct and standardize addresses in over 240 countries. Eliminate returns, cut postage expenses, prevent fraud and keep your customers happy by verifying their address before you send a package.

• Reduce address correction fees – save up to $10 per package
• Efficiently validate and correct addresses every time you ship
• Maintain high customer satisfaction

Accurate data. Delivered.

✓ Address Verification	✓ IP Location
✓ ID Verification	✓ Name Parsing
✓ Email Verfication	✓ Phone Verification
✓ GeoCoding	✓ Record Matching

www.MelissaData.com/global
or call 1-800-MELISSA (634-4772)

MELISSA DATA®
Your Partner in Data Quality

On premise or remote APIs available

ORACLE USER GROUP LEADERS' SUMMIT

Several Java user group (JUG) leaders participated in the Oracle User Group Leaders' Summit January 14–16, 2013, at Oracle headquarters. The International Oracle User Group Community (IOUC) is a community of leaders representing Oracle users groups worldwide. More than 100 user group leaders attended the summit to learn about Oracle products and technologies, provide feedback to product groups, network, and share best practices.

In the dedicated Java track, Java experts from Oracle presented the current state and roadmaps for Java SE, JavaFX, Java EE, Java ME, and Oracle Java Embedded. Java Evangelist **Arun Gupta** discussed the features in the upcoming Java EE 7 release. **Mike Lehmann,**

Counterclockwise from top left: Mike Lehmann (left) and Arun Gupta; Ryan Cuprak (left) and Ben Evans; Bruno Souza; Stijn Van den Enden (left), Johan Vos, and Bert Ertman

PHOTOGRAPHS BY MARGOT HARTFORD

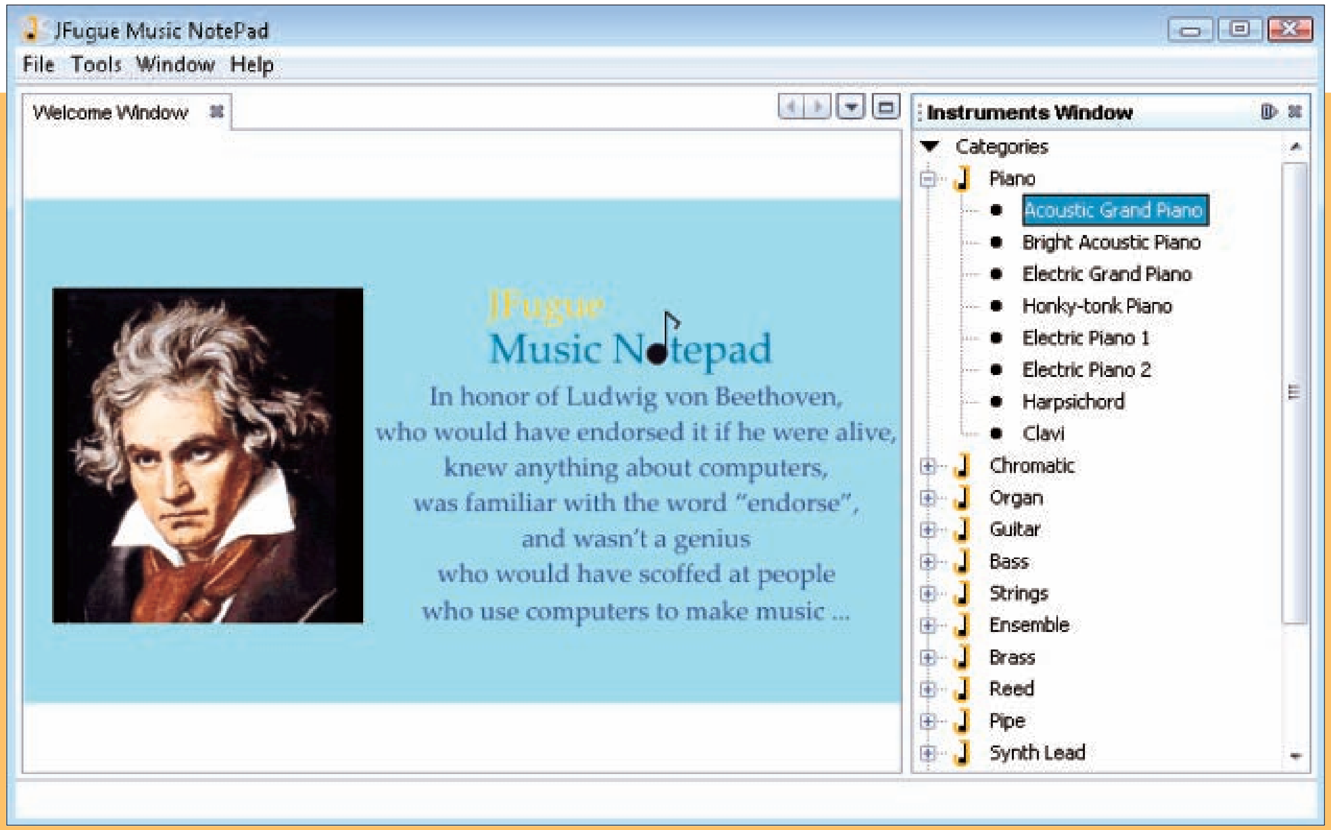
senior director of product management at Oracle, presented an overview of the Oracle Java Cloud Service offering and gave a demo. There were also sessions on JUGs and the Java Community Process (JCP), Java 8, JavaFX, embedded Java, Java.net, Oracle Technology Network, and *Java Magazine*. Throughout the sessions, JUG leaders were encouraged to provide feedback, and they weren't shy, offering many suggestions about features, processes, and programs.

At the session on the JUGs and the JCP, JCP Executive Committee member **Bruno Souza** spoke about the Adopt-a-JSR program. "The biggest job in Adopt-a-JSR is education. Many people say, 'I want to help . . . what's the JCP?'" Even small actions can have a big impact, and contributors don't have to be experts, he said.

Jim Bethancourt, Houston JUG leader, was a first-time summit attendee. "This has been a great experience," he said. "I've got a lot of good information to take back to my JUG and also my company." **Bert Ertman**, leader of the Netherlands JUG, comes to the summit as much for the networking as the sessions. "It's great to spend time with other JUG leaders and share ideas about fostering community growth and participation," he said.

—Tori Wieldt

Sharat Chander discusses the summit.



In honor of Ludwig von Beethoven,
who would have endorsed it if he were alive,
knew anything about computers,
was familiar with the word "endorse",
and wasn't a genius
who would have scoffed at people
who use computers to make music ...

FEATURED JAVA.NET PROJECT

JFUGUE MUSIC NOTEPAD

Geertjan Wielenga leads the development of Java.net's JFugue Music NotePad project. The software, which is a NetBeans Platform application, provides a graphical user interface to JFugue, an open source Java API for music programming. JFugue Music NotePad is one among several projects that use the JFugue API.

JFugue Music NotePad lets users create and edit music using visualized music notation. Because of the simplicity of the JFugue system, even people without formal music training can experiment with making music using JFugue's easily understandable notation scheme. Furthermore, the hardware and software complexities of MIDI are bypassed.

This video provides everything you need to get started with JFugue Music NotePad, from downloading, to opening, to running, to understanding the project structure.

If you are interested in contributing to the JFugue Music NotePad project, visit the project page on Java.net.

Java Developers Give with Kiva

Kiva is a global community that endeavors to empower people in less advanced nations by providing individuals with microloans (as little as US$25), to help them start or build a business. Java developers are participating in Kiva through an initiative started by Java Champion **Mattias Karlsson** (pictured), the "Java Users Around the World" Kiva Lending Team. The team's stated reason for lending: "We strongly believe that even small things could change the world. 'Be the change you want to see in the world!'—Gandhi."

Since the team was founded in May 2009, the 113 team members have provided 920 loans totaling US$25,675 to entrepreneurs in Africa, Asia, South America, the former Soviet republics, Mongolia, and elsewhere. The loans can be for any business that interests the lender. If you'd like to participate, consider joining the Kiva Lending Team.

JAVA CHAMPION PROFILE

HARSHAD OAK

***Harshad Oak** is a developer, author, Java Champion, and the founder of IndicThreads and Rightrix Solutions.*

Java Magazine: When and how did you first become interested in computers and programming?

Oak: I don't remember being excited about computers in the pre-internet era. But the internet opened a whole new world. So while in college, I explored the internet, toyed with HTML, and wrote freelance tech articles for mainstream magazines and newspapers. I studied programming much later on, during my master's course in computers.

Java Magazine: What was your first professional programming job?

Oak: My college placed me at i-flex solutions, where my first project was a Java EE–based e-payment solution. I have been with Java EE since. Curiously enough, i-flex was acquired by Oracle a few years after I left and is now Oracle Financial Services Software Ltd. So I am in a way ex-Oracle.

Java Magazine: What happens on your typical day off?

Oak: Spending time with my four-year-old son would be top of the list. I also spend a lot of my free time on social causes promoting scientific temperament and citizen participation in governance. I am also currently working on a nonprofit initiative at Sudhar.in. *Sudhar* means *reform/improve* in the local language. Apart from that, I like to attend talks and discussions on various tech, non-tech, and social issues. Listening to people who have dedicated their lives to an idea or a cause is perhaps the fastest way to enrich oneself.

Java Magazine: Has being a Java Champion changed anything for you with respect to your daily life?

Oak: The Java Champion accolade certainly provides many opportunities to do more. On a personal level, the Java Champion tag is catchy enough for even my aunts and uncles to amuse themselves by calling out to me as "Java Champion" instead of my name.

Java Magazine: What are you looking forward to in the coming year?

Oak: I am currently working on my fourth book. This one is about Java and the cloud and will be out later this year. On the business front, we hope to take the IndicThreads brand further and do more to foster and facilitate tech communities and tech enterprise in India.

Find more about Oak at HarshadOak.com.

JAVA USER GROUP PROFILE

BRASILIA JAVA USERS GROUP (DFJUG)

The Brasilia Java Users Group (DFJUG) was founded in February 1998 by **Daniel deOliveira**, a founding Java Champion. DFJUG is the oldest Java user group (JUG) in Brazil. Each of the 26 most active JUGs in Brazil has a particular area of focus and DFJUG's is teaching Java to beginning programmers.

For many years, DFJUG has offered free Java courses in classrooms contributed by universities. The JUG also trains blind and deaf people who would like to become Java programmers. Brazilian law requires companies with more than 100 employees to employ disabled people at a rate of at least 5 percent.

Software companies and the developers themselves greatly welcome the training that DFJUG provides. On Java.net, DFJUG has created a Java mobile app, Rybena, that enables communication between the blind and the deaf through a cell phone.

The JUG is also active in bringing software engineering educational materials to the Portuguese-speaking developer community and has published 22 programming books in Portuguese.

In October 2006, the JUG became the leader of the new Brazilian Java Education and Development Initiative (JEDI-BR). Dozens of volunteers spent six months translating Java instructional materials into Portuguese and these are used in an online training program, which has provided Java training to thousands of developers throughout the Portuguese-speaking world.

"We know that we are making a difference in these people's lives," deOliveira says. "More than 41,000 programmers trained for free."

NetBeans IDE 7.3 Supports HTML5

Now available, NetBeans IDE 7.3 introduces HTML5 capabilities for creating and debugging Web and mobile applications, while continuing to provide its proven support for traditional desktop, mobile, and enterprise application development. New HTML5 features in the NetBeans IDE include an HTML5 project type, a list of HTML5 samples, an enhanced HTML editor that offers code completion for new HTML5 elements, a new JavaScript editor and debugger based on the Nashorn JavaScript project, a CSS rule editor, and an embedded browser based on the WebKit open source browser engine.

NetBeans IDE 7.3 is bundled with a Chrome extension that provides smooth interaction between the NetBeans IDE and the Chrome browser, providing bidirectional interactions between NetBeans and Chrome.

NetBeans IDE 7.3 includes tools for developing HTML5 applications based on the JavaScript Backbone.js library, from the Java API for RESTful Web Services (JAX-RS), which in turn can be generated from databases registered in the IDE. The Java Editor has been enhanced in a number of ways, including new breadcrumb support, clipboard history, and several new hints and code suggestions that help to identify bugs early in the development cycle. NetBeans IDE 7.3 also includes enhanced support for a range of technologies including Java EE, JavaFX, Groovy, PHP, and C/C++.

CODE RECOMMENDERS

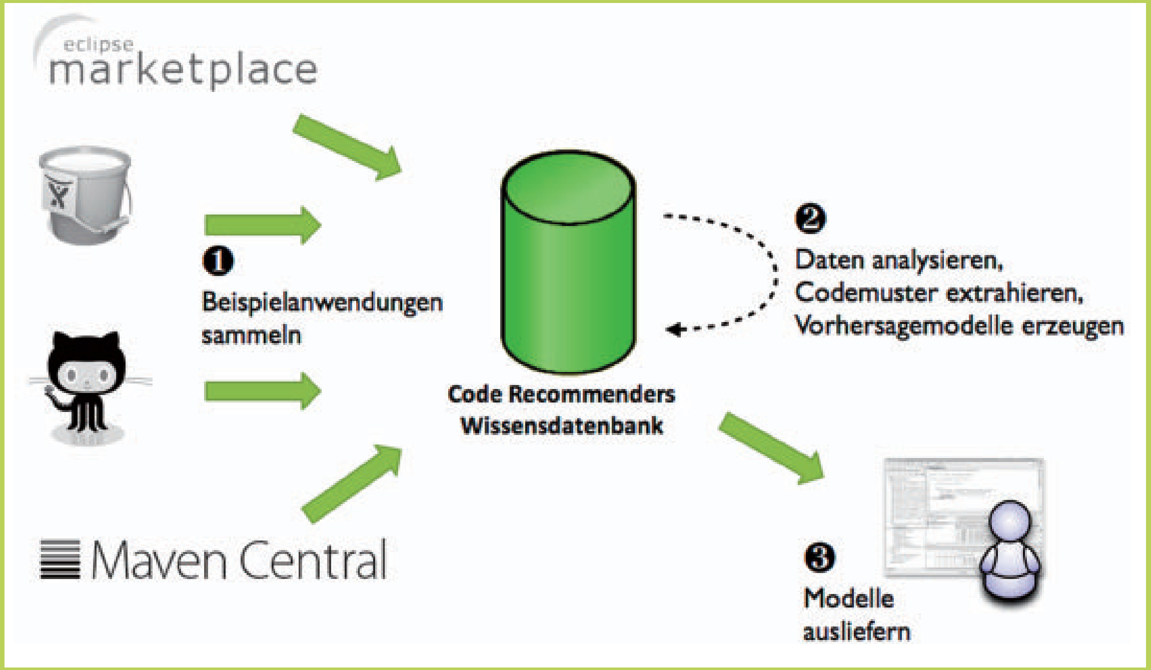
Code Recommenders is a new open source module being developed by members of the Eclipse Foundation. The project, led by **Marcel Bruch**, utilizes the vast Java codebase that has been made available to the Eclipse Foundation by the global Java developer community as a database for predicting the keystrokes a developer is most likely to enter next. In other words, this is a new form of intelligent code completion: one based on data mining and source code syntax analytics.

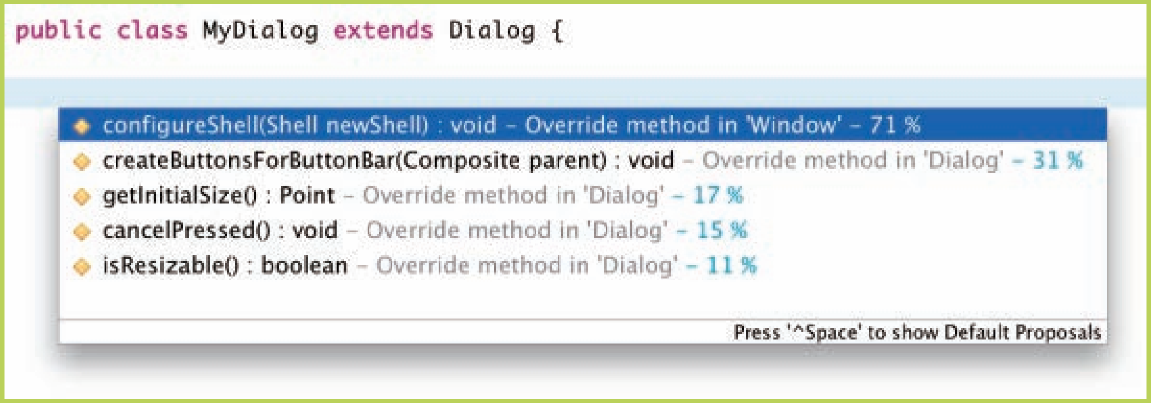
Bruch uses Amazon.com to illustrate the principle: "You bought some books. Amazon finds the set of customers who bought the same books, and then suggests other books that might interest you, based on the additional books those other customers purchased.

"Code Recommenders does something similar, only with Java code. It analyzes the code you're currently entering, and then looks for code snippets in the Eclipse Java code repository that follow the same syntactic pattern. It analyzes what the developers most frequently entered next in that coding context, and provides you with suggestions based on that analysis."

So, instead of being presented with an alphabetized list of possibly hundreds of methods that can be selected to complete a statement within a given coding context, Code Recommenders provides the developer first with the methods that were used most by other developers, sorting all the possibilities based on the percent of lines of source code that used each method.

Code Recommenders: intelligent code completion based on Java source code contributed by the global Java community.


eclipse marketplace
❶ Beispielanwendungen sammeln
Maven Central
Code Recommenders Wissensdatenbank
❷ Daten analysieren, Codemuster extrahieren, Vorhersagemodelle erzeugen
❸ Modelle ausliefern



```
public class MyDialog extends Dialog {
```

```
configureShell(Shell newShell) : void – Override method in 'Window' – 71 %
createButtonsForButtonBar(Composite parent) : void – Override method in 'Dialog' – 31 %
getInitialSize() : Point – Override method in 'Dialog' – 17 %
cancelPressed() : void – Override method in 'Dialog' – 15 %
isResizable() : boolean – Override method in 'Dialog' – 11 %
                                        Press '^Space' to show Default Proposals
```

Top: An overview screen shows the process of going from code to recommendations. Bottom: A screen displays recommendations for an SWT text widget in code completion.

EVENTS

JavaOne Russia *APRIL 23–24*
MOSCOW, RUSSIA

Attend JavaOne Russia for two intense days of Java-based content, training, and community networking. Thousands of community experts attend every year and share their insights on Java. The agenda is packed with keynotes, technical sessions, hands-on labs, demos, exhibitions, and more. The tracks include Core Java Platform; JavaFX and Rich User Experiences; Java EE, Web Services, and the Cloud; Java ME, Oracle Java Embedded, and Java Card; and Oracle Java Embedded for Business.

PHOTOGRAPH BY GETTY IMAGES

APRIL

Oracle CloudWorld
APRIL 2, MUMBAI, INDIA
APRIL 4, SINGAPORE
APRIL 9, TOKYO, JAPAN
ADDITIONAL DATES PLANNED FOR ENGLAND, GERMANY, MEXICO, AND THE US
Explore the Oracle cloud in one day in a city near you. Mobile, social, and cloud are redefining how business gets done. Discover what your company can do to take advantage of these new opportunities and gain a competitive advantage.

JAX 2013
APRIL 22–26
MAINZ, GERMANY
This conference on Java and the enterprise focuses on current and future trends in Web, Android, software architecture, cloud, agile management methods, big data, and more. More than 170 speakers will deliver 200 sessions, and 25 topic-focused daylong sessions are offered.

QCon
APRIL 25–27, BEIJING, CHINA
APRIL 28–29, CHENGDU, CHINA
QCon is the enterprise software development conference designed for team leads, architects, and project management and is organized by the community, for the community.

MAY

JavaOne India
MAY 8–9
HYDERABAD, INDIA
Don't miss JavaOne India for two intense days of Java-based content, training, and community networking. See the JavaOne Russia description for details.

Great Indian Developer Summit (GIDS)/Java
MAY 10
BANGALORE, INDIA
The Java track at GIDS gives equal emphasis to Java and other languages such as Clojure, Groovy, JRuby, and Scala that run on the Java Virtual Machine. Oracle's Arun Gupta is a speaker.

Geecon 2013
MAY 15–17
KRAKOW, POLAND
With the slogan "let's move the Java world," the conference focuses on Java-based technologies, dynamic languages, rich internet application enterprise architectures, patterns, and more.

JEEConf
MAY 24–25
KIEV, UKRAINE
This annual conference brings together those who use Java technologies for application development. The focus is on modern approaches in development of

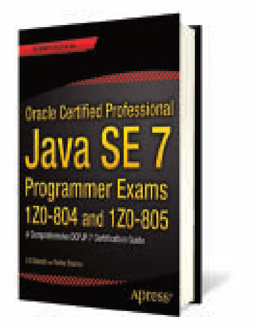
distributed, highly loaded, scalable, enterprise systems with Java; innovations and new directions; interesting architectural decisions based on Java technologies; integration with other languages, tools, and libraries to develop modern applications; and popular directions and trends in the world of Java development.

JUNE

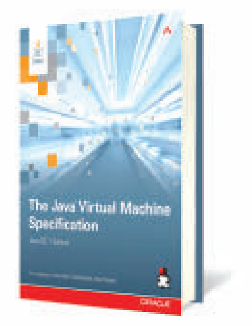
JAXConf 2013
JUNE 3–5
SANTA CLARA, CALIFORNIA
JAXConf 2013 is a comprehensive deep-dive event for software and enterprise development professionals in the modern Java ecosystem. JAXConf gives developers, architects, and project leads the opportunity to hear about the latest technical and methodology developments driving the software development community forward.

QCon New York 2013
JUNE 10–14
NEW YORK, NEW YORK
QCon aims to empower software development by facilitating the spread of knowledge and innovation in the enterprise software development community. This practitioner-driven conference is designed for team leads, architects, project managers, and engineering directors.

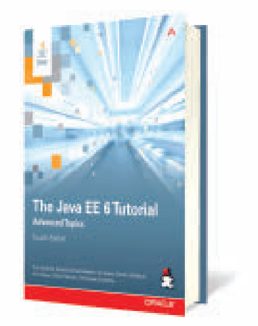
JAVA BOOKS



ORACLE CERTIFIED PROFESSIONAL JAVA SE 7 PROGRAMMER EXAMS 1Z0-804 AND 1Z0-805
By S G Ganesh and Tushar Sharma
Apress (March 2013)
Oracle Certified Professional Java SE 7 Programmer Exams 1Z0-804 and 1Z0-805 is a concise, comprehensive, step-by-step, and one-stop guide for the OCPJP SE 7 Exam, and is meant for anyone studying for the OCPJP certification. This book has a comprehensive focus on all 13 exam topics rolled out by Oracle and maps exam topics to the book chapters. It includes two mock tests, an instant refresher that summarizes the most-important concepts, and an API quick reference that covers the most-important classes and methods relevant to the exam topics.



THE JAVA VIRTUAL MACHINE SPECIFICATION, JAVA SE 7 EDITION
By Tim Lindholm, Frank Yellin, Gilad Bracha, and Alex Buckley
InformIT (February 2013)
This reference guide provides complete, accurate, and detailed coverage of the Java Virtual Machine in the Java SE 7 platform. It fully describes the invoke dynamic instruction and the method handle mechanism introduced in Java SE 7, and the formal Prolog specification of the type-checking verifier introduced in Java SE 6. The book includes the class file extensions in Java SE 5.0 for generics and annotations, and aligns the instruction set with the Java Memory Model from JSR 133. The authors also clarify many aspects of linking and initialization.



THE JAVA EE 6 TUTORIAL: ADVANCED TOPICS, FOURTH EDITION
By Eric Jendrock, Ricardo Cervera-Navarro, Ian Evans, Devika Gollapudi, Kim Haase, William Markito, and Chinmayee Srivathsa
InformIT (January 2013)
This example-driven, task-oriented guide to developing enterprise applications for Java EE 6 provides new and intermediate Java programmers with a deep understanding of the platform. This guide, which builds on the concepts introduced in *The Java EE 6 Tutorial: Basic Concepts, Fourth Edition*, contains detailed introductions to more-complex platform features and instructions for using the latest version of the NetBeans IDE and GlassFish Server Open Source Edition.

JAVA IS COMMUNITY

Java and community go hand in hand. The stronger the Java community, the better the Java language. Innovation is only possible when Java developers around the world voice their ideas.

If you're new to Java, or to the community participation, there are many ways to "start small." For those of you who have been swimming in the Java waters for a long time, maybe it's time to step up to the next level. Opportunities for participation range from simply joining a Java user group (JUG) to launching the next big Java event—and more. Being part of the Java community is a great way to meet enthusiastic people and to share your passion.

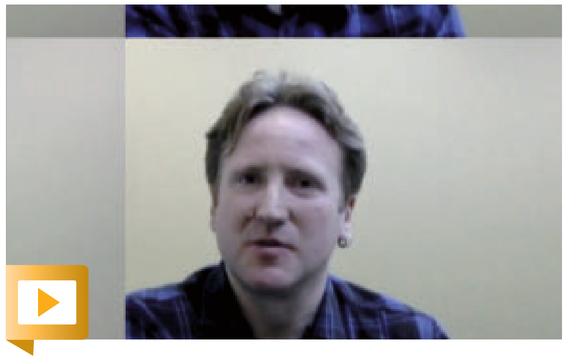
In this hands-on guide, **Java community leaders offer their advice and experience on 13 topics.**

Whatever you do—participate! Java will be better with your involvement.

—Caroline Kvitka and Agnès Crepet

ART BY I-HUA CHEN

1 START A JUG



Ben Evans explains how easy it is to start your own JUG.

If your city is in need of a JUG, consider starting one. JUGs provide a meeting place for Java users to get information; share resources and solutions; increase networking; expand Java technology expertise; and most of all, have fun. When establishing your JUG, consider how factors such as geography, education, economy, and culture make it unique, and try to address those factors.

HERE ARE SOME QUICK TIPS TO GET YOU STARTED:

01 **Add your JUG** to the JUG map and get listed as a Java.net project.

02 **Get involved** in the JUG leaders community by joining their mailing list and participating in the monthly JUG leaders call.

03 **Ask all members** to spread the word about your new group.

04 **Find a university** or local IT company to sponsor a meeting location, and then invite the university students or company employees to join your JUG.

05 **If there is a local Oracle office** in your area, make contact and let them know about your JUG.

06 **Establish a regular meeting time,** and have senior programmers in your community give talks on interesting topics.

07 **Utilize the technical content** available on Oracle Technology Network and *Java Magazine*.

08 **Consider inviting local headhunters** or recruiting companies to your meetings. Some JUGs allow recruiters to pass out their cards in exchange for a small donation (US$25–US$50) for refreshments.

09 **Post your meeting** and events dates on the Java.net JUGs Community page. This keeps Oracle and potential speakers aware of your events, just in case they have an engineer or evangelist in (or traveling to) that area.

10 **Contact your local universities,** particularly instructors of Java courses, and ask them to post your meeting notice on public boards. Offer tutoring help to students who become members.

11 **Contact publishers** of Java books. Some will send you books to pass out during meetings. Publishing book reviews on your site will help. Check the summarized list of JUG sponsorship programs.

12 **Avoid politics** between sponsors and headhunting firms. Give equal opportunities to all sponsors.

13 **Establish a good Website.** A flexible CMS system can make your JUG site more interesting.

Mauritius Java User Group is one of the newest JUGs to be started. The Republic of Mauritius is an island nation in the Indian Ocean.

14 **Establish a service** to inform JUG members about job vacancies.

15 **Be online**—answer e-mails, tweets, and instant messages quickly.

16 **Keep the core leadership team small**—two to four people who are passionate about Java and the JUG.

17 **Attend the biggest Java conferences** (such as JavaOne and Devoxx) and the Oracle User Group Leaders' Summit. You will meet other JUG leaders and the Oracle team and find potential presenters.

—**Ahmed Hashim**, Egyptian Java Users Group founder

2 JOIN A JUG

Nichole Scott shares the benefits of JUG membership.

3 RUN A HACK DAY

The best way to get day-to-day developers involved in Java standards and the OpenJDK is simply to get them hacking on code. Hack days are a key part of the growing momentum within the Adopt-a-JSR and Adopt OpenJDK programs. In the London Java Community (LJC), we've been helping to improve Java through our hack day program and we've learned a lot on the way.

In order to run a hack day, you need only one thing: desire. The goal is simply to try out the latest API or do something useful with OpenJDK.

Anyone involved in the Adopt-a-JSR or Adopt OpenJDK program should consider running a hack day for an area they are involved in. JUGs are a good place to run hack days, because they have direct access to developers and ties to the wider Java community. Further, if your company is considering adopting or already relies on one of these technologies, you could run a hack day to experiment with it.

Open source project communities are another ideal group to run hack days. An existing codebase can be cleaned up, or a community can be brought together. There is also an opportunity to give back to core Java by testing your project on future OpenJDK or Java EE versions.

HERE ARE SOME TIPS FOR RUNNING A SUCCESSFUL HACK DAY:

01 **Establish goals.** What do you and others expect to get out of the hack day? You can choose to educate and inform, send feedback to the standards committee, find bugs, fix bugs, or simply have fun. Working out goals up front with the spec lead/Expert Group is a good idea.

02 **Get a venue and set a time.** These events are best held face-to-face. Your venue can be almost anywhere. In the LJC, we've had no difficulty getting companies to offer space, but hack days can be run inside any community space, a coffee shop, or a living room. Weekday

hack events are suitable for small API checks, while weekend events are useful for more-complex topics.

03 **Communicate.** Talk to both attendees and the people involved in the technology itself. For an Adopt-a-JSR hack day, talk to the Expert Group of the relevant JSR. For an Adopt OpenJDK hack day, follow the guidelines of finding a project sponsor and submitting patches to the GitHub project. You also need to set expectations with your attendees. Give a brief introductory talk: tell them why they're there and what's expected of them.

04 **Expect a wide range of experience.** If you're running an Adopt-a-JSR hack day, remember that the JSR will be used by people of mixed experience. Your group should reflect this.

05 **Consider creating exercises.** When running a JSR 310 hack day, we created exercises in the form of failing unit tests that people had to make pass. People enjoy the challenge of an exercise if it is done well.

—Richard Warburton, London Java Community

Nety Herawaty (left) and Mila Yuliani teach students about Java while on a tour in Malang, Indonesia.

4 TAKE JAVA ON THE ROAD

In October 2012, we went on the road via motorcycle in Malang, Indonesia, to promote Java and to check on the progress of Java Education Network Indonesia (JENI). JENI is an integrative curriculum for students in Indonesia to learn, share, and develop Java-based solutions. It was created by JUG Indonesia and was officially adopted by the Ministry of National Education in 2006. We wanted to find out how many vocational high schools in Malang were teaching Java to software engineering majors, and how many schools had implemented JENI in their curriculum.

Meruvian, a nonprofit organization located in Jakarta that focuses on Java and open source, covered our transportation costs and expenses. This road show was also a Duchess Indonesia event.

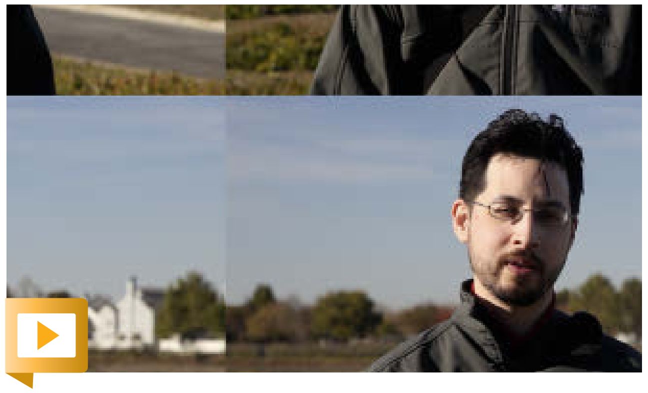
During the road show, we visited 32 vocational high schools in Malang. If a school was not teaching Java, we met with a teacher to assess whether there was interest in implementing Java or JENI in the school's curriculum. If the school was interested, we provided a free day of Java training. Oracle supported our efforts by providing *Java Magazine* postcards, Java 7 T-shirts, and Duke dolls.

We also introduced students to a Meruvian program called jTechnopreneur, a technology entrepreneurship program. This program allows vocational high school students to do a one-year internship and then to continue the jTechnopreneur program as a professional or entrepreneur after graduation.

Of the 32 high schools we visited, 22 have IT majors and 9 have software engineering majors. Of these, 5 were already teaching Java to their students and 2 had implemented JENI.

We're planning another road show to vocational high schools in Jepara, Indonesia, and other cities. We will also introduce junior high school students to Java using Greenfoot—playing games with Greenfoot will pique their interest in Java.

—Nety Herawaty and Mila Yuliani, Duchess Indonesia


Stephen Chin shares Java road tour tips.

5 LAUNCH AN EVENT

Is your country or region in need of a Java event? If you are considering launching a new Java conference, the following are some keys to success, based on my experience launching the JMaghreb conference in Morocco.

01 **Attend a large Java conference** such as Devoxx, JavaOne, or Jfokus to see how the event works and make connections.

02 **Set realistic expectations.** Expecting your inaugural event to be huge can lead to failure.

03 **Set a date** several months away that is available in the Java conferences calendar.

04 **Select a venue** that will support the expected number of attendees. If funds are an issue, scout out a free location at a university or a company.

05 **Find the right contact** at each potential sponsor.

06 **Prepare a thorough partnership package,** send it out early to your sponsors list, and follow up.

07 **Stay neutral** and give equal opportunities to all the sponsors.

08 **Build an event steering committee** made up of people who are passionate about Java.

09 **Create a conference Website** with compelling content and contact information.

10 **Open a call for papers** and spread the word.

11 **Make the registration form** and process easy.

12 **Create social media accounts** and keep your followers informed about new sponsors, confirmed speakers, and sessions.

13 **Be online.** Answer e-mails quickly and be interactive in your social media accounts.

14 **Post your conference on Websites** such as Java.net and Lanyrd.com and publications such as *Java Magazine* and Oracle's *Java Developer Newsletter*, to keep potential speakers and attendees aware of your conference.

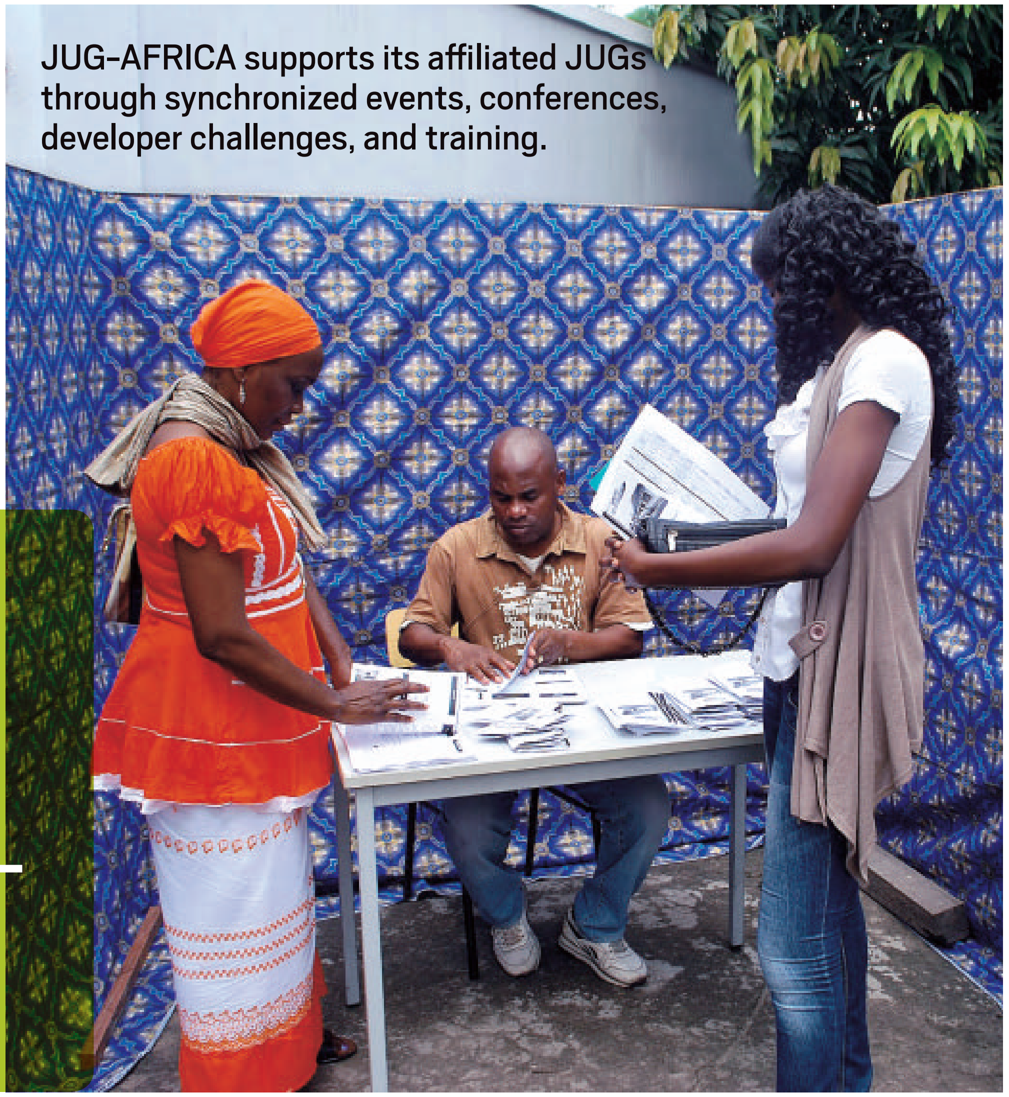
15 **Send a reminder** to your registered attendees before the conference date.

16 **Make your event special.** Include local flavor or some kind of surprise.

17 **Following your event,** send a survey to attendees and seek feedback from speakers and sponsors. Send sponsors pictures, "best of" videos, session survey forms, and the attendee contact list (if that was previously agreed on).

—**Badr El Houari,** MoroccoJUG Leader and JMaghreb Manager

6 START A REGIONAL JUG


JUG-AFRICA supports its affiliated JUGs through synchronized events, conferences, developer challenges, and training.

If starting a regional JUG is on your radar, I can offer you my experience in launching JUG-AFRICA. JUG-AFRICA is an umbrella JUG for the entire continent with which individual JUGs can affiliate. More than 20 JUGs have already affiliated, totaling 6,000 members from 18 countries.

JUG-AFRICA promotes communication among JUGs located in Africa to allow them to collaborate globally in ways that will ultimately benefit Java developers locally. Individual JUGs continue to function normally, and affiliation does not subordinate local JUGs. JUG-AFRICA exists solely to support the affiliated JUGs by connecting its members and providing services to ease challenges linked to the regional context such as cross-cultural communication and coordination, getting speakers and sponsors, and accessing technical resources.

JUG-AFRICA fulfills its mission by organizing large-scale programs focusing on Java and its ecosystem that have an impact on the whole continent. These programs include synchronized events, regional conferences, developer challenges, and online training.

Imagine thousands of developers from all around Africa with the same focus at the very same time—that's what synchronized events are all about. JUG-AFRICA acts as a liaison to obtain sponsorships and fosters collaboration between individual JUGs to organize local events around specific themes. These events allow local participants to be involved in regional initiatives and are a great opportunity for local speakers to shine. For example, Java 7 launch events were held in 11 cities over the course of one week.

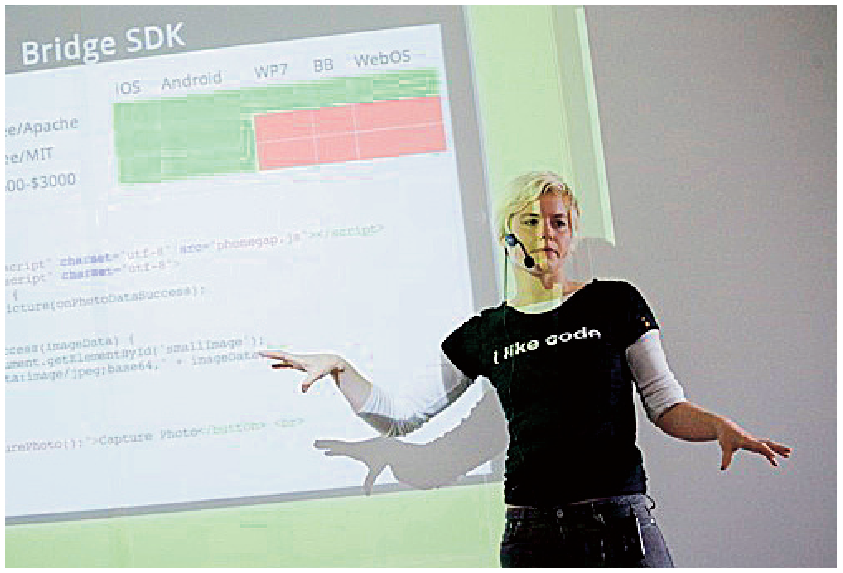
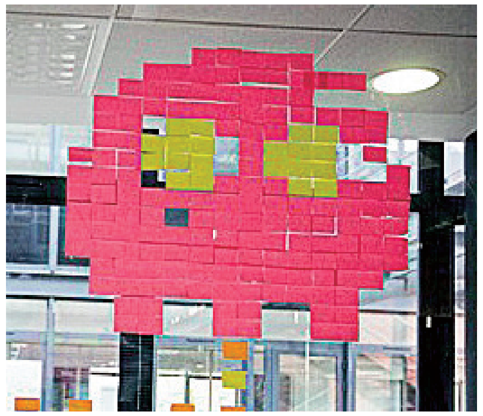
Regional conferences give JUG members the opportunity to get together and network in person. These events also have more of an international flavor, with participants and speakers coming from all around the world. For example, JCertif 2012 in Brazzaville, Republic of the Congo, featured speakers from the US, Canada, France, the UK, Togo, the Democratic Republic of the Congo, Cameroon, and Burkina Faso. Regional conferences are a great way to showcase the vibrancy of the African Java community, as they attract a wider audience with attendees ranging from businesspeople and journalists to students and researchers.

Developer challenges are designed to motivate and encourage African developers to create innovative applications driven by and for local content. JUG-AFRICA's role is to obtain sponsorship and prizes for the event and manage communications and advertising for the event on the international scene. In each country, local JUGs publicize the event to encourage entries, select judges, and organize events such as hackathons and prize-giving ceremonies. Developer challenges are another great way to showcase African talent, as contestants are guaranteed to gain exposure throughout the continent and beyond.

JUG-AFRICA organizes online Java training sessions. Twice a year, nearly 50 participants enroll in a 10-week training program that allows beginners to get started in Java development and more-experienced developers to hone their skills. These courses always feature the latest updates or versions: for example, the sessions starting in May 2013 will use Java 8. This gives JUG members access to technological innovations.

—Max Bonbhel, Founder and President, JUG-AFRICA

Did You Know?
Duke is open source. Duke fans are free to give a personal touch to the original Java mascot. On Nov. 13, 2006, Sun announced that Duke would become free graphics, just as the implementations of Java ME and Java SE became free software.

The Lyon JUG keeps members engaged through a speaker academy and by holding its own event, MIX-IT, focused on agile and Java.

PHOTOGRAPHS BY MAXIME COUTURIER

7 REINVIGORATE YOUR JUG

I lead two JUGs in France: the Lyon JUG and Duchess France (a global network for women in Java). For a long time, the main activity of the Lyon JUG was to organize conferences. Once a month, we would invite a speaker to talk. The JUG members didn't participate much during these events. Recently we revitalized our events by involving more people from the local Java community. We launched a "speaker academy" to train people to give talks. We now suggest lightning talks in our meetings, to be given by speaker academy participants at an upcoming meeting. Our JUG leaders and experienced speakers in the community coach the speaker candidates. Their first lightning talks might be a little stressful for them, but taking the plunge can lead them to speaking opportunities at bigger events.

I reinvigorate my user groups by inviting the local Java community to propose hands-on sessions. The attendees help each other and share knowledge.

Another way to boost your JUG is to produce a podcast. We launched our podcast, Cast-IT, with the Lyon JUG team last year for those who want to speak about their favorite framework or tool. Many people are more comfortable being behind a microphone rather than on stage. It's another way for people to share their passion for software craftsmanship.

—Agnès Crepet, Lyon JUG and Duchess France

8 SPEAK AT JAVAONE

Trisha Gee tells you how to start speaking at events, from your JUG meeting to JavaOne.

Gail Anderson (top right) and Paul Anderson (bottom right) at JavaOne in Brazil.

Paul Anderson and I are currently developing course materials for JavaFX and writing a book on JavaFX with the NetBeans platform. We felt that our topic was technologically worthy, so we submitted a proposal for JavaOne that appealed to both prospective attendees and the conference reviewers.

The title, abstract, and takeaways (session learning objectives) are the three fields attendees read to decide whether to attend your talk. There is no magic formula for creating an accepted submission, but a catchy title helps. This is your chance to entice people to attend your session. Our title, "Make Your Clients Richer: JavaFX and the NetBeans Platform," was not only catchy but also accurate. The abstract should provide enough detail so attendees understand what you plan to cover. The learning objectives should be concrete skills or how-to's that attendees can apply in their own work.

Make these fields as clear and interesting as possible. One of the best ways to do this is by brainstorming possible titles and content with others. Is the title clear? Does it pique my interest? Does it make me want to read the abstract? After reading the abstract, would I sign up for the session? Are the learning objectives skills I can use in my own work?

The JavaOne session submission process also includes a field that reviewers use to further evaluate your submission. This is your chance to explain the technical merits of your subject matter to the reviewers, why your submission is valuable to the community, and why attendees would have an interest in attending.

The Algeria JUG held its first meeting on Software Freedom Day 2011.

If your proposal is accepted, my next advice is simple: practice, practice, practice. This will ensure that your session material fits into your allotted time slot and that you're comfortable navigating through any demos. You should also make sure that your session is organized and flows well. Switching between slides and live demos helps keep your audience engaged.

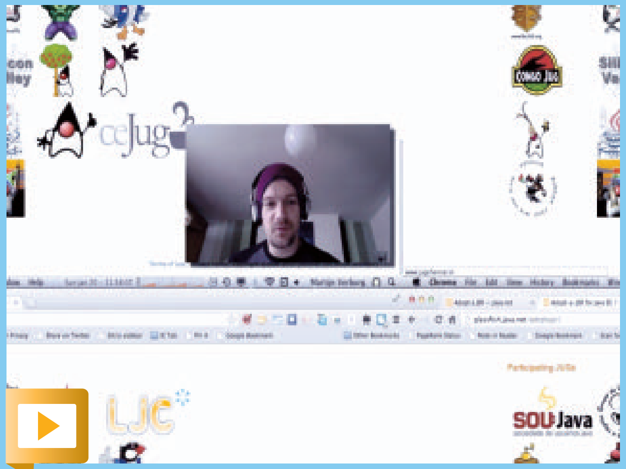
When we presented at JavaOne Latin America, our talk was simultaneously translated into Portuguese. With live translations, it becomes even more important to speak clearly and not too rapidly.

You will meet key Java community members at the conference. Pursue these meetings to touch base with colleagues and make new contacts. Stay in touch by contributing material through blogging, e-mail, online forums, or Twitter.

—**Gail Anderson,** cofounder of the Anderson Software Group

With JUG Cologne on board with JSR 346, the number of JUGs participating in the Adopt-a-JSR program has now grown to 13.

9 ADOPT A JSR



Change the world, one JSR at a time. Martijn Verburg of the London Java Community shows you how to get started with the Adopt-a-JSR program. All Java developers are welcome to bring their skills and enthusiasm to this global effort.

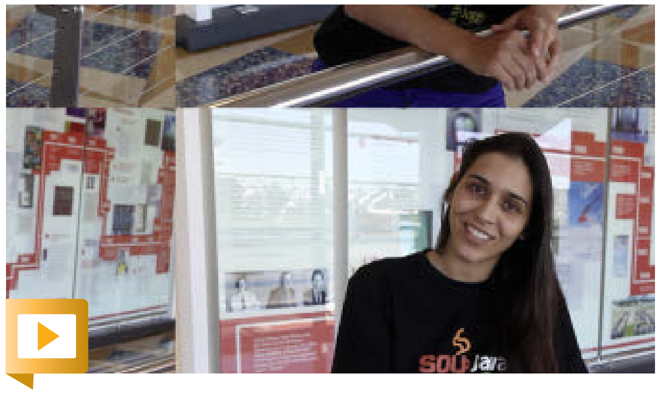
10 BECOME A JAVA CHAMPION

The Java Champions are an exclusive group of passionate Java technology and community leaders who are nominated and selected by the Java Champion community.

Members come from a broad cross-section of the Java community. They include Java luminaries, senior developers, architects, and consultants; academics; authors of Java-related content and industry conference speakers; and JUG leaders and the managers of Java-related portals.

The Java Champions program exists to recognize those who make significant contributions to the Java ecosystem; to work to protect the health of the Java ecosystem; and to provide a point of outreach to the broader Java community. The group helps guide the Java ecosystem to keep it as open, relevant, and fun as possible.

Who makes a good candidate? Java Champions are leaders and technical luminaries. They are independent-minded and credible. Java Champions are involved with some really cool applications of Java technology or some humanitarian or educational effort; the application must be openly available to the Java community.


Yara Senger shares her path to becoming a Java Champion.

Moreover, Java Champions are able to evangelize or influence other developers through their own professional activities.

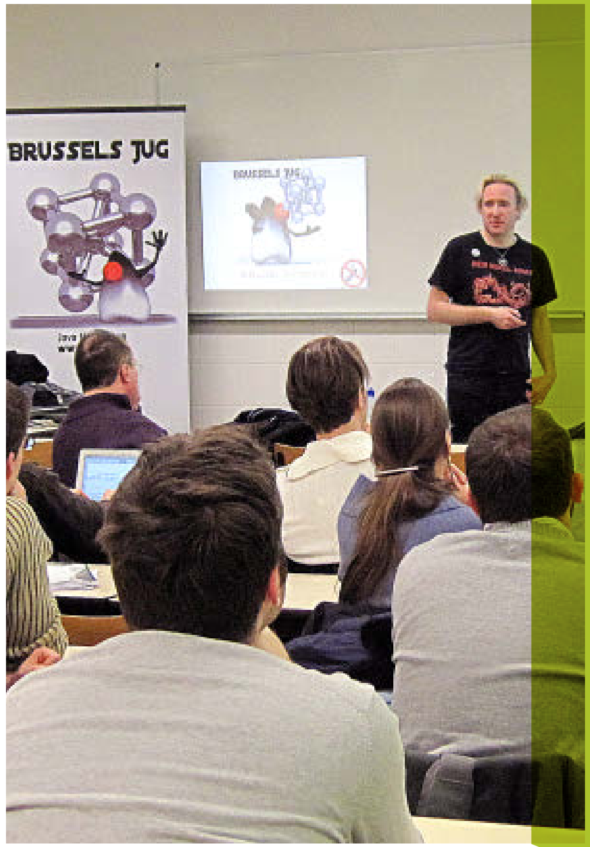
Java Champion nominees are named and selected through a peer review process. The candidate's contributions to the Java community must be available to its members either online or in readily available print media. The nomination must cite specific examples (such as URLs, book titles, or presentations) of the candidate's direct contributions in order for the candidate to be considered by the selection committee. To nominate a candidate, follow the guidelines on the Java Champions page.

The Java Champions are an independent group that uses a peer review process to select new members. A committee of Java Champions review qualified nominations and make recommendations to the group. The group must reach consensus to accept a nomination.

If you care about the global Java ecosystem, then consider nominating those who you think are leading lights.

—Martijn Verburg (aka "The Diabolical Developer"), on behalf of the Java Champions community

Members of the Belgium JUG and Brussels JUG gathered for a joint event.



11 MANAGE A BIG JUG

Managing a JUG community of several hundred members can be a demanding task. On one side, there are the logistics of organizing activities. On the other side is the challenge of "the psychology of the community."

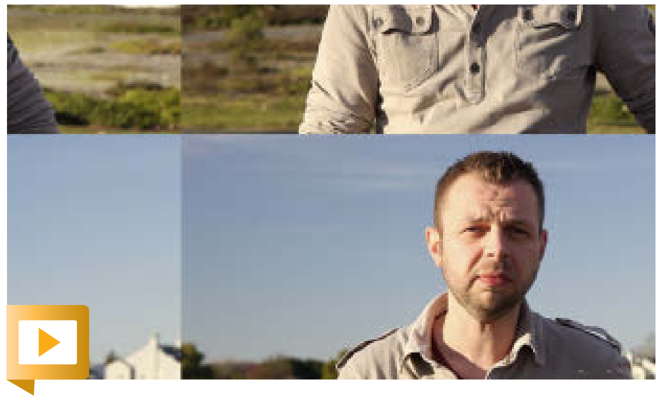
Logistical challenges include finding speakers; conference venues; and, ultimately, sponsors. To attract speakers, having a good network can help. The most important rule is don't be afraid to ask—even if the speaker is a rock star. It gets your name out there.

Locating suitable conference venues and finding sponsors are connected—you need support from the industry to find cool places. Identify the companies that have an interest in being known by your community. Prepare a professional sponsor handout that lays out the costs and short- and long-term benefits.

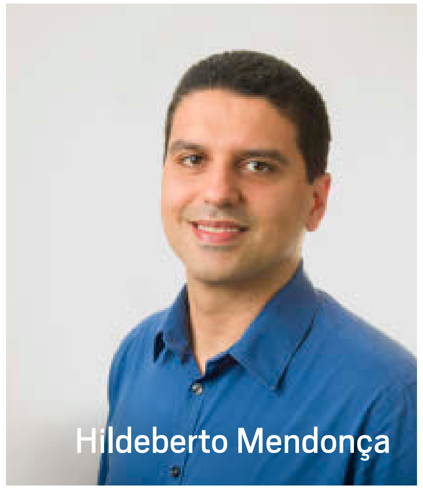
Logistics can be managed, but the community itself is more of a challenge. Choosing the right conference topics can be tricky. Niche topics might interest fewer community members; however, if you choose mainstream topics, then you are just mainstream. Remember to take your members' family obligations, weariness, and hunger into account when planning after-work activities.

The most important thing that a community needs is drivers who stand up and "do." To quote entrepreneur Derek Sivers, "The first follower is what transforms a lone nut into a leader." The followers will show up when the lone nut enters the scene—don't be afraid to enter the scene.

—Tasha Carl, Brussels JUG


Bert Ertman offers advice on managing a big JUG.

12 LEAD A JUG



Java Magazine talked with three JUG leaders about their experiences in starting, growing, and leading a JUG. **Bruno Souza** is leader of SouJava, **Michael Hüttermann** is leader of JUG Cologne, and **Hildeberto Mendonça** is leader of CEJUG—The Ceará Java User Group.

Java Magazine: Why did you decide to start/join/lead a JUG?

Souza: In 1998, we had a group of developers that had been meeting monthly for two years to discuss and learn Java. Thinking we could invite others to participate, we decided to form an official JUG, and SouJava was born.

Hüttermann: It was over 10 years ago. I'd moved to Cologne, Germany, and was

BRUNO SOUZA PHOTOGRAPHED BY BOB ADLER; HILDEBERTO MENDONÇA PHOTOGRAPHED BY TON HENDRIKS

wondering why such a big city with so many IT companies and developers did not have an active, organized JUG. So I started a small JUG. We now have a very vital community.
Mendonça: In 1998, when I first saw the potential of Java Applets, I got addicted to developing charts. From then on, Java became my professional programming language. I wanted to learn and teach Java all the time until the point when I was doing it for the masses. Leading a JUG was just a natural step.
Java Magazine: When starting a JUG, what obstacles might someone face?
Souza: Regardless of how enthusiastic they are, people come and go. If the JUG is not constantly forming new leaders and organizers, the group can disappear. Promote and empower your members, so they can become leaders when their time comes. Invite active participation.
Hüttermann: If you start a JUG, then you definitely need a lot of stamina. It might start slowly. It's a process, so don't give up. Also, organizing a JUG takes a time commitment. With JUG Cologne, many people wanted to actively support the group but couldn't commit the time.
Mendonça: The main challenge is to keep motivation and enthusiasm up all the time. For that, we engage as many people as possible in several different activities. It's also very important to involve members in all relevant decisions.
Java Magazine: Once a JUG has been started, it needs to acquire a core membership, and some corporate sponsorship is also quite beneficial. How do you sustain and grow a JUG in relation to these areas?
Souza: Fulfill a need in your area; that will bring members and sponsors. Solve a local issue facing developers and their companies. Use your prestige and influence as JUG leader to improve the local ecosystem. Groups get sponsorship from software vendors, but also look beyond these to companies in the local ecosystem: training companies, English schools, recruiting firms, and so on. We've even seen a bakery sponsor the coffee break at a JUG event. Be creative.
Hüttermann: In Cologne, actually, we're a lightweight, loosely coupled group. Companies support us by freely providing locations or catering. Regarding the membership: there's a vital core, and dedicated working groups are organized by their own hosts. Community is about giving and taking, and normally, you give more than you take.
Mendonça: We've learned that the more knowledge we share, the more mature and stable the group becomes. When members experience collective problem solving, they get deeply attached to the group. We clearly state to sponsors that spending money on knowledge sharing is one of the fastest ROIs they can get.

Duke was originally created by Joe Palrang to be the "agent" for the Green Project at Sun Microsystems. Duke became the Java mascot when Java technology was first announced, around the same time that the first Java cup logo was commissioned.

13
ADOPT OPENJDK

Ben Evans of the London Java Community explains how you can get involved in the Adopt OpenJDK project.

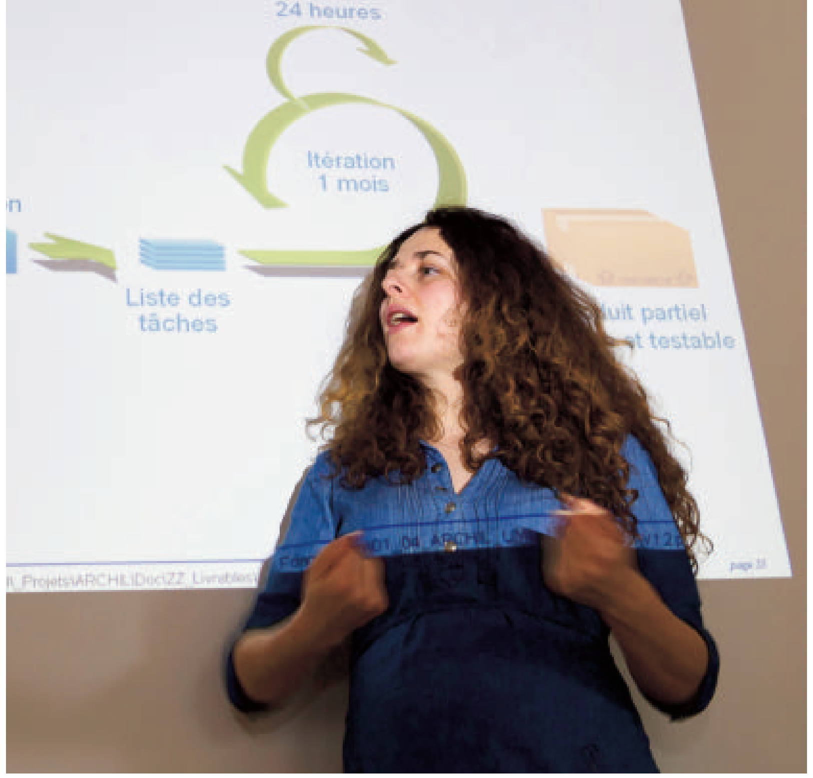
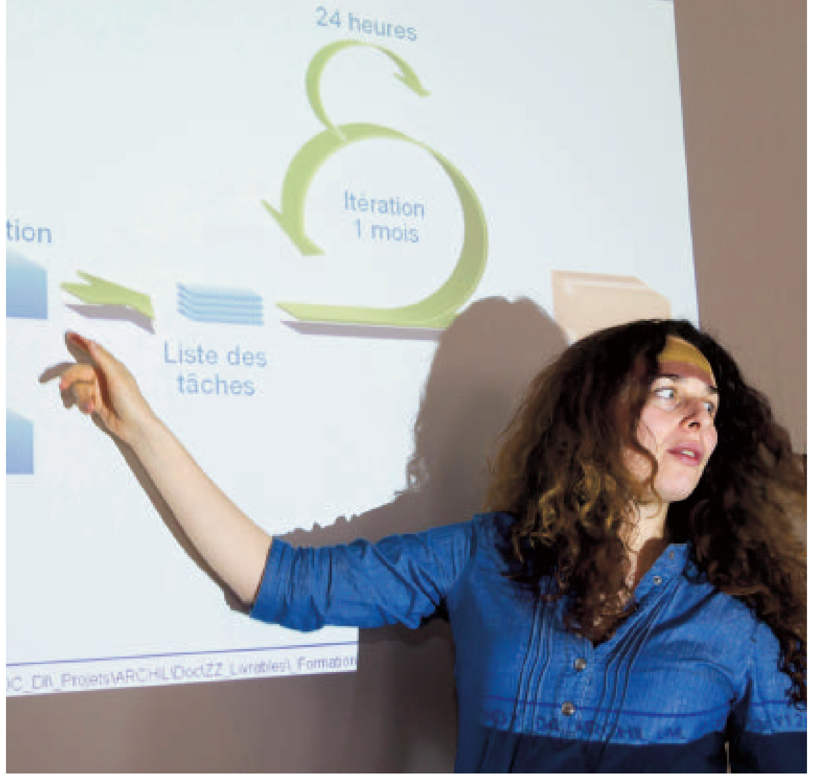
Meet Our Guest Editor

Q&A WITH AGNÈS CREPET

BY KEVIN FARNHAM

Agnès Crepet is an entrepreneurial Java enterprise architect, a Java Champion, and a leader of Duchess France and the Lyon Java User Group. The editors of Java Magazine chose her as guest editor of the community issue because she embodies the spirit of today's strong Java community leaders. We were inspired by her passion for building community, her adventurous nature, and her eagerness to help others learn Java. As guest editor, Crepet added insight into the making of our community how-to cover story and wrote the section on reinvigorating your Java user group (JUG).

PHOTOGRAPHY BY TON HENDRIKS

Crepet discusses the iterative cycle of agile software methodologies, which she uses at her day job as a Java architect for Boiron, a French pharmaceutical company that specializes in homeopathy.

Java Magazine: You recently took a sabbatical from work. Can you tell us about that?

Crepet: After working for 10 years in France, I decided in 2011 to take a sabbatical with my boyfriend, who's a Java senior developer. We wanted to not only discover beautiful landscapes but also meet Java communities all over the world, because both of us are active in the Java community and passionate about it.

We started a world tour, and our first stop was in Africa. We didn't want to be tourists staying in a hotel—we wanted to meet African people. A friend of mine told us about a consulting company seeking to invest in training young West African engineers. This gave us an opportunity to work in Africa. We spent five weeks doing volunteer work training Java students in Togo.

The head of the company explained that many young African people dream of leaving for more economically advanced nations, such as Europe or the US. Hoping to give them the desire and pride to stay in their own country, he devised this project to create new opportunities for developers in Africa—and it fit in perfectly with our own travel objectives.

We were seeking the same kind of experience in our visit to Asia: to avoid hotels and to meet real people. We wanted to connect with experimental organizations that are already engaged in educational projects around Java. We crossed Malaysia, stopping in Kuala Lumpur to meet the JUG, and then contacted the Indonesian school Meruvian, a nonprofit organization focused on Java and open source in Jakarta. Meruvian's promoters teach computer science and Java to young people between 16 and 21 years old. It was a wonderful experience, and we met awesome people—especially the young women who have launched Duchess Indonesia.

Java Magazine: During your trip, you were involved in launching Duchess Africa and Duchess Indonesia, chapters of Duchess, the global network that connects women in Java technology. What difficulties did you encounter in establishing these groups?

Crepet: In Africa, it was not very easy to launch a Duchess group because there were not many women among our students: only two, but they are motivated. In Africa, few girls go to school, let alone have access to scientific studies, so there are very few female computer engineers. Moreover, these women were beginners in Java, so they were a little bit stressed about launching a group dedicated to Java development. But after a few weeks, they became more confident and we launched Duchess Africa.

In Indonesia, it was easier because

MAKING REAL CONNECTIONS
"[We wanted] to avoid hotels and to meet real people . . . to connect with experimental organizations that are already engaged in educational projects around Java."

in the Meruvian school a lot of women are involved. I started to correspond with some women from Meruvian, who explained that they were very motivated about launching a Duchess Indonesia group. They managed everything efficiently, building Websites, making a video, and organizing a great Duchess Indonesia launch event at Gunadarma University with around 200 attendees.

Java Magazine: Do you have any updates on their progress?

Crepet: Duchess Africa is now a small but motivated group. Duchess Indonesia leaders Nety Herawaty and Mila Yuliani did a road show project to promote Java last October in Malang [East Java, Indonesia], visiting 32 vocational high schools one by one on motorcycle, and teaching Java. They are awesome women.

Java Magazine: You visited many African JUGs. What do you see happening there?

Crepet: JUGs and the Java conferences they sponsor are a critical element in the growth of the technology sectors of African nations. African developers get involved in these communities, perhaps more than us, and generate dynamism because they are passionate about them.

Java Magazine: And now, a few personal-profile questions. Where did you grow up?

Crepet: I grew up in Saint-Étienne, near Lyon, France. Saint-Étienne is saddled with a reputation as a gray, unfashionable working-class city, but I like its authenticity.

Java Magazine: When and how did you first become interested in computers and programming?

Crepet: There was no computer in the family. But during my primary school, I discovered BASIC. Later, some friends introduced me to Linux and the world of free software.

Java Magazine: What were your first computer and programming language?

Crepet: My first computer was a cheap one (Intel with the Debian OS), and my first programming language was C (I started computer science via the artificial intelligence domain).

Java Magazine: What was your first professional programming job?

Crepet: I was a Java developer for a banking publisher.

Java Magazine: What do you enjoy for fun and relaxation?

Crepet: Organizing gigs and cultural events through my association, Avataria.

Java Magazine: Has being a Java Champion changed anything for you with respect to your daily life?

Crepet: Now, people from all over the world ask me to train them in Java.

Java Magazine: What are you looking forward to in the coming years?

Crepet: My world tour changed my point of view. In Africa and in Asia, I met greatly motivated people. I think these communities will grow in the coming years, and their dynamism will inspire us. Regarding women in computing, something similar could happen. At my Indonesia talks, I was surprised that more than 50 percent of the attendees were women. In Europe, only 5 or 10 percent of attendees at conferences such as Devoxx are women. We should follow the example of Indonesia.

You can find out more about Agnès Crepet at Ninja Squad, the company she started to promote software craftsmanship; Mix-IT conference, the Java/agile conference she founded; and Cast-IT, a podcast site. Follow her on Twitter (@agnes_crepet). </article>

Kevin Farnham is the editor of Java.net and a regular contributor to *Java Magazine*'s Java Nation section. He is also the owner of Lyra Technical Systems, a small consulting and publishing company, through which he works on software engineering projects involving mathematical modeling and simulation, and scientific data analysis.

LEARN MORE ABOUT AGNÈS CREPET

- Ninja Squad
- Mix-IT conference
- Cast-IT podcast
- Twitter

JCP Executive Series

A Conversation with Steve Harris

CloudBees' **Steve Harris** addresses the need of the JCP to adjust to open source and the cloud. **BY JANICE J. HEISS**

PHOTOGRAPHY BY PHIL SALTONSTALL

Continuing our series of interviews with distinguished members of the Executive Committee of the Java Community Process (JCP), we turn to Steve Harris, senior vice president of products at CloudBees. Harris has a long history of working with Java at the cutting edge. After initially leading a Smalltalk-based startup that was bought by ParcPlace-Digitalk, where he became vice president of engineering, he joined Oracle in 1998 to manage a team that delivered native Java support inside the Oracle database. He eventually became Oracle's senior vice president of application server development, where he was in charge of product development for Java EE, before joining CloudBees in September 2011.

CloudBees, a self-described Java platform-as-a-service (PaaS) company committed to the idea that the cloud is the new platform, aspires "to free developers from infrastructure maintenance duties so they can focus 100 percent on developing great applications." The cloud changes the Java platform and the way developers work. We discuss this below with Harris and get his views on how those changes need to be reflected in the JCP.

CloudBees was recently elected to the Executive Committee of the JCP, with Harris as its representative.

Left: Steve Harris gets a progress report from Mark Prichard, senior director of product management at CloudBees. Right: Harris prepares for a presentation.

Java Magazine: In your JCP Position Statement you say, "Let's move the JCP toward an as-a-service model of helping the community deliver value and away from being a process-bound handmaiden of Oracle." What specific changes might accomplish this?

Harris: The JCP and the Executive Committee have, I think, traditionally operated more as a gate to progress where they approve a certain path of development. Instead of being a gate to future activities, the JCP needs to figure out how to be an organization that's helping Java and Java developers progress as a community. CloudBees itself offers a PaaS—hosted services for Java developers where they pay on demand by the minute. So as an organization, the JCP needs to be operating in a way that's helping the Java community. It should principally be driven by developers.

Java Magazine: You are quoted as saying, "My objective is to work within the Executive Committee to find ways to relax controls and lower barriers for involvement and community contribution so that the JCP serves as a facilitator for driving the Java platform forward." What specific controls need to be relaxed?

Harris: There's been a lot of work in the JCP over the last year to try to make it more open and transparent. The work to date has been accomplished without changing the formal governing rules of the JCP itself, though. There are the rules under which the JCP works—the Java Specification Participation Agreement [JSPA]—that individuals and companies sign in order to participate. It's a big document that's really complicated and makes you feel like you want to consult your lawyer.

So part of the challenge for the Executive Committee is to revise the way the JCP works so that it becomes easier to participate. One of the things under consideration is to let individuals participate under an easier set of rules so that individual developers can review documents and comment on them without having to sign five-page legal agreements with their employer's consent. The work is underway now in JSR 358, with the entire Executive Committee as members.

Java Magazine: CloudBees seems to have a strong sense of urgency about how the Java platform needs to change. Where does this urgency come from?

Harris: CloudBees is a company that delivers a hosted Java PaaS—essentially everything a Java developer needs to create a new application, test it, stage it, and make sure it's working properly to deploy it into production.

Harpreet Singh, senior director of product management at CloudBees, updates Harris on his latest project.

There are no salespeople between you as a developer and making that happen. You don't have to install any software. It's just there. In the cloud.

The Java platform has historically been about how you interact with a specific machine to create a piece of software that will run on that machine. With the cloud, instead of interacting with a specific machine to create a piece of software that lives on it, you actually need to interact with the cloud, which is some sort of hosted environment that can be scaled out to many different processors. The Java platform needs to adapt to this new cloud model. The Java platform has historically been able to adapt to different models—for example, it's still the best place to build service-oriented applications. It has evolved over time. But it faces the big challenge of addressing the cloud and the way people are now working using cloud-based resources.

The sense of urgency comes about because this is happening *today*. CloudBees as a company is driving how Java developers interact with the cloud. But ultimately, we want some standardized means of doing that. So your investment in working with what's at CloudBees translates to other options in the cloud, including Oracle or large traditional vendors, as well as other startups. That's a lot of what Java is about: write once, run anywhere. We believe that "Develop continuously, deploy continuously anywhere in the cloud" is the model moving forward. The Java platform needs to address this through the JCP.

ON CLOUD

"[The Java platform] faces the big challenge of addressing the cloud and the way people are now working using cloud-based resources."

Java Magazine: You've said that the JCP needs to find ways to match its value-add to the way developers are working, not the other way around. What is the JCP missing about the way developers are working?

Harris: Most though by no means all Java innovation takes place in open source communities—places like the Eclipse Foundation, and more ad hoc forums where source code is shared—social coding forums. GitHub is a great example of that. So that's how people are working today. They're not sitting down in committees, deciding on the next iteration of the technology and how to standardize it, which is the model that the JCP was originally built around. The JCP needs to reflect the importance of open source communities. I think that, with some rethinking, this is possible. For example, the restrictions that the JCP places on the creation of Reference Implementations, specifications, and tests needs to be better mapped to the way that people work in open source communities.

The Java developer community is used to being able to just say, "OK. I'm putting something together and licensing it." Or, being able to pick a favorite license out of many. And this license specifies, for example, that if you use what I produced, you won't sue me over patented IP within it. These things are formalized in open source licenses already. But the open source licenses and the JCP don't map well—the JCP operates differently. The JCP Executive Committee is working on ways to change this by revising the JSPA. Java developers would probably prefer to just have a license and then be left alone.

People who participate in the JCP have, I think, an expectation that they are doing it to share their intellectual property in some fashion. They want people to use the specification and the Reference Implementation and the tests that they put together. And they want to do it in a way that then encourages compliance. That's the expectation that people have walking in. And it's true of companies, too. IBM, for example, is leading a batch JSR to standardize how you schedule batch processes in Java. They want everyone to use that. They are contributing their intellectual property to encourage broad uptake. So at its heart, I think the participants expect their intellectual property to be used broadly.

But there are real intellectual property problems that companies can have. So for example, let's say we want to include in the next Java version something that will support multi-touch capabilities on devices. I should be able to write code in Java that lets me do multi-touch stuff. We all know that Apple has patents that cover this area. Yet you want to develop something against a Java standard that works and does multi-touch. Can you, and will you get sued over it? This is the kind of example where vendor interests and the JCP intersect in challenging ways. The JCP is really obligated to pay attention to the intellectual property issues and how they flow, who has rights to do what, and so on.

ON COMMUNITY
"The JCP needs to figure out how to be an organization that's helping Java and Java developers progress as a community."

Java Magazine: CloudBees has made a contribution to the Java community by making milestone OpenJDK builds available and supporting FOSS [free and open source software] projects. Tell us about this.

Harris: There are two parts. OpenJDK has been very successful as an open source project. That's reflected in a standard that's defined through the JCP. OpenJDK is being driven as an open source project with involvement from Oracle, IBM, Red Hat, and many others. The Java platform is evolving through that open source project as a work in progress. As they progress, they produce milestone builds. Everyone wants people to try out the builds and report any issues. By allowing these milestone builds to be broadly consumable by developers, then you get broader uptake and improve the final product.

CloudBees has a free hosted service that allows people to develop new applications using Java, and we've made those milestone builds available to people to try. So if you have a Java application you built and you want to try it on a new OpenJDK milestone that has just been introduced, it's very easy. You can try it out against the application you built. And if you see a problem, then you can let the JDK project folks know. It makes it easy for developers to provide feedback on the OpenJDK as it progresses. CloudBees has a program where if you are a developer and you're creating a free open source project, then we make CloudBees resources available to you free of charge up to a certain scope. It is a way for us to be more engaged with open source communities.

Java Magazine: Tell us about the difference between an open source process and the standards process of the JCP. And how does the cloud fit in?

Harris: There is an impedance mismatch between a standards process and an open source development process. In open source development, you're building a piece of software, making changes to it, and people can try it out. You're going through multiple revisions as you move toward something that's more production quality. Even after it's released, you're always making changes to it, improving it. And people are consuming it constantly in some fashion.

In a standards process, which is at the heart of the JCP, you're after producing a specification that captures what this technology does, a Reference Implementation—in other words, something that actually implements the specification—and a set of tests that validate that any implementation you create conforms to the specification. Pulling those three things together, which is the task of Expert Groups, takes a long time. It's a big effort. There is a mismatch between the instant gratification enjoyed in open source projects and the committee process within the JCP.

ON PROCESS

"There is a mismatch between the instant gratification enjoyed in open source projects and the committee process within the JCP."

With the cloud model, an application uses, consumes, and exposes services that are hosted in the cloud. For example, the Java platform hosted in the cloud by CloudBees can be used by any developer. You can also use other hosted services with your application on CloudBees, through what we call a partner ecosystem. In our world, nobody is delivering packaged software, just hosted services. Partners provide testing services, code quality services, monitoring services, and so on. Those are hosted services in the cloud. Instead of installing a piece of software on your machine to help you monitor your application, for example, you just say, "I want to use the service that monitors my application," and you click a button to use it. It's a different model from what the JCP was created for. We need to ask ourselves how the JCP-traditional specification, plus Reference Implementation, plus compliance test suite, should change to reflect this new hosted service model.

Java Magazine: What is your assessment of recent changes at the JCP?

Harris: First, some recent changes within the JCP rules requiring committees to operate transparently have helped. People can now observe what's going on and comment without signing up and sitting in committee meetings. And the introduction of the Java user groups as participants in the Executive Committee has been incredibly positive. The London JUG has done great work. They've created programs like Adopt a JSR and so on that encourage developers to read what's going on in an Expert Group and participate. This makes individual developers more involved with what's going on and gives them a voice.

And recently, there are greater efforts to encourage individuals and companies to try out and test things as they emerge through the JCP. All of this increases engagement among developers. If the JCP ceased to exist tomorrow, most Java developers would wonder why they should care—but eventually, they would care. When they found that there was fragmentation with a particular piece of Java technology that mattered to them, then they would actually care quite a bit. Frankly, the mechanics of how these things are accomplished within the JCP are pretty remote to most developers. Improving the connection developers have with the actual creation of the definition and direction of the Java platform is really important. </article>

Janice J. Heiss is the Java acquisitions editor at Oracle and a technology editor at *Java Magazine*.

LEARN MORE

- Java Community Process
- CloudBees

Paul Perrone shows off the trunk of smarts in “Tommy Jr.,” a fully autonomous self-driving robotic car that was a semifinalist in the DARPA Urban Challenge.

JAVA IN ROBOTICS

Java pioneer **Paul Perrone** creates real-time frameworks for sensing, measurement, and control.

BY DAVID BAUM

PHOTOGRAPHY BY DAVID DEAL

Perrone Robotics Founder and CEO Paul Perrone amid just-unpacked boxes at the company's new headquarters in Charlottesville, Virginia

SNAPSHOT

PERRONE ROBOTICS
perronerobotics.com

Headquarters:
Charlottesville, Virginia

Industry:
High technology

Employees:
15–20

Java technologies used:
Java Platform, Standard Edition 1.6 and 1.7, with a possible Java ME Connected Limited Device Configuration (CLDC) profile on the horizon

What do an art gallery, a shoe store, and a 1959 Lincoln Continental have in common? All of these items are touched by the rapidly expanding world of Paul Perrone, an entrepreneur in the field of Java-based robotics systems.

Perrone is well known in the Java community as an architect, author, and speaker on Java, Java EE, and XML. His work with embedded Java utilizes advanced sensor technology such as lasers for everything from triggering in-store ads and protecting artwork to guiding vehicles such as LincVolt, a retrofitted Lincoln Continental owned by rock star Neil Young, who spearheaded its creation.

Perrone started Assured Technologies in 1998 and later founded Perrone Robotics, a provider of software for robotics and automation, where he currently serves as CEO. He is a Java Champion and the recipient of three Duke's Choice Awards, including a Golden Duke and a Lifetime Achievement Award. He is a frequent presenter at JavaOne conferences and a former Java user group leader, and he's authored several books on enterprise software technology, including *Building Java Enterprise Systems with J2EE* (Sams, 2000) and the *J2EE Developer's Handbook* (Sams, 2003).

In 2001, Perrone started developing a large-scale distributed communications framework for the robotics industry called MAX, a general-purpose platform designed for a variety of commercial, military, consumer, and professional robotics and automation applications. Since then Perrone Robotics has used MAX to develop self-driving robotic cars, unmanned air vehicles, factory and roadside automation applications, and a wide range of advanced security applications.

"We did some research to better understand where robotics was going. We saw a lot of interesting projections and trends that pointed to an explosion in the market," Perrone recalls. "Previously, robotic applications utilized stovepipe architectures, where everything was created from scratch. While there were several university programs based on open source projects, there was clearly a lack of standards and no robust robotics framework that could be practically used."

As a result, implementing even moderately complex robotics and automation solutions has historically been tremendously expensive. Developers generally have to start from scratch on each new project. They mostly use specialized hardware and software to create monolithic applications that are difficult to extend and costly to deploy.

Perrone decided to remedy this problem by creating MAX—favoring Java for this task because of its ability to dynamically upload and update robots with new logic and new code.

"Java was a natural fit because of its basis as a high-level programming language," Perrone explains. "It is object-oriented, has a lot of built-in APIs, works with many third-party tools, and is supported by a large developer network. Java is still growing and is one of the most popular programming languages in the world."

MAX is growing in tandem as Perrone and his team create frameworks for a variety of tasks. One MAX extension contains software drivers for reading, processing, and controlling laser data. Another MAX framework handles vehicle measurement, with general-purpose libraries of objects such as Vehicle, Motorcycle, and Tractor Trailer. This library permits applications to use sensor data to measure the characteristics of vehicles, such as their height, weight, length, and speed. Java makes it easy to integrate different types of sensors into these advanced measurement systems.

PLATFORM NEEDED
"There are open source platforms in the robotics world that people expect will organically grow. I think that's the wrong approach for robotics."

"There are open source platforms in the robotics world that people expect will organically grow," Perrone notes. "I think that's the wrong approach for robotics. A robot with arms or legs or wheels can pose a real threat, which merits a formal platform."

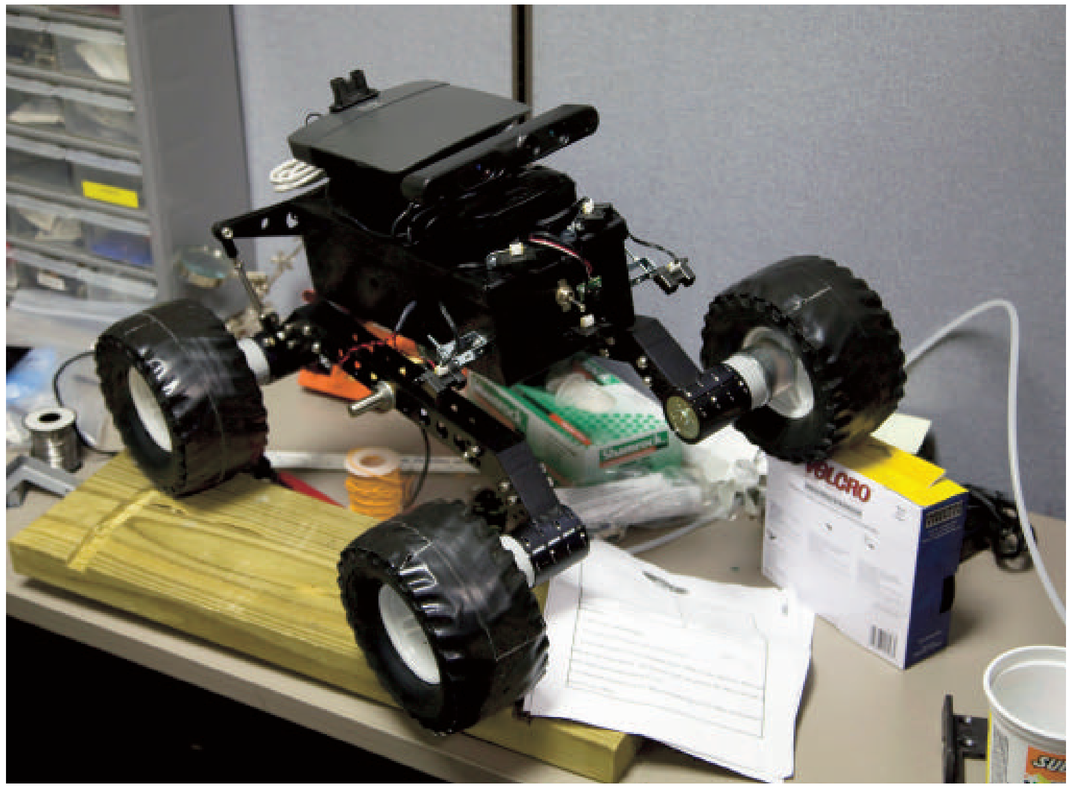
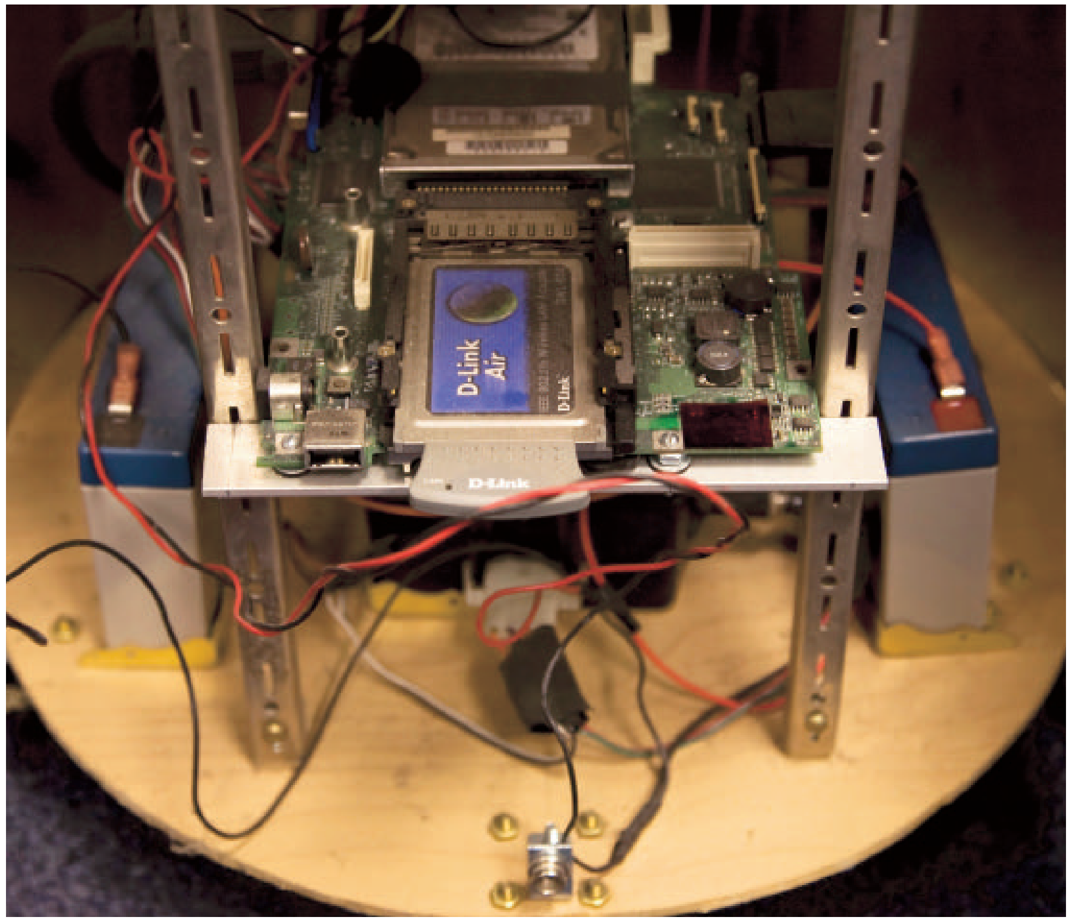
Perrone developed a "healthy paranoia," as he puts it, for safety when he created and tested control systems for railways and trains. "That was about the time Java started appearing on our radar and we started to use it in these systems," he notes. "We didn't want to just create something and throw it over the fence in open source form, then later feel responsible for a mishap that sets robotics back 10 years. We don't need another AI [artificial intelligence] winter like they had in the 1980s."

THE ROAD TO ROBOTICS

Perrone studied computer engineering as an undergraduate at Rutgers University and then went into the PhD program at the University of Virginia. After his university research efforts, he went to work for Harris Corporation, which later became a joint venture with General Electric. He quickly became interested in AI, expert systems, and rule-based programming to solve complex problems.

In 2005 Perrone was the leader of a team that advanced in the Defense Advanced Research Projects Agency (DARPA) Grand Challenge, a historic race of self-driving vehicles across the Mojave Desert. His team invested about US$60,000 and 10 man-months in the effort, yet their entry competed favorably against multimillion-dollar, multiyear projects. "Our goal was to demonstrate how we could do things faster and cheaper with the MAX platform," he states.

Perrone's team entered again in the 2007 DARPA Urban Challenge with another self-driving autonomous car, this time navigating the vehicle through a city landscape. The team's vehicle used laser technology to calibrate its distance from adjacent objects, drawing a virtual map of its surroundings.

Top: Rumbles, an ARM-based security-bot vehicle; bottom: the inner workings of Beaker, the first MAX-based robot

Rumbles takes a ride around Paul Perrone's house.

Perrone Robotics used similar technology, a Java Real-Time System (Java RTS), for the Pennsylvania Turnpike that measures vehicles in motion. Laser-triggered cameras above the roadway can identify motorcycles, cars, trucks, and tractor trailers, gathering data on vehicle type, height, width, length, and speed. The data is combined with lane-control data and transmitted via Ethernet to a vehicle scanning station server, assisting with everything from law enforcement to traffic control. In combination with in-ground scales, the system eliminates the need for trucks to pull off the road into a weigh station.

"We liked the simplicity of the Java language and the fact that you could use high-level tools to rapidly generate code," says Perrone, reflecting on these interrelated projects. "We also liked Java's object orientation and support for third-party tools. All that made it an attractive thing to work with. That's what lured me in at the time."

Perrone especially liked Java's inherent portability. The platform was designed with the internet in mind, using applets to transfer data and logic in the form of functional, executional code. "I knew that would be useful in robotics," he says, "since a lot of behavior can be realized just by specifying things in configuration files without writing new code. Java is relatively simple and both hardware- and OS-independent," he adds. "There is a wide range of built-in, commercial, and open source tools readily available [to developers], which makes it an attractive and low-cost platform for developing robotics applications."

FROM SECURITY TO MARKETING TO FACTORY AUTOMATION

Soon after the DARPA challenges, Perrone started experimenting with ways to apply MAX to the domain of security. He devised the concept of "security walls" by using lasers to create invisible fields that react when broken. For example, art museums can create security walls that sound an alarm if somebody reaches to touch a painting. Retail stores can use them to keep people from walking into a stock room or approaching a cash register. Factories can use them to keep people from entering restricted or hazardous areas. A derivative application, dubbed Laser Tag, uses security zones in conjunction with RFID tags to allow authorized personnel to enter these spaces without triggering the alarms.

The technology can be used for advertising and marketing as well. For example, a shoe store could mount a TV display above its running shoes, then play a short video that describes the merits of each product when consumers pick up the merchandise.

"All of these applications have similarities in that they are analyzing and detecting patterns from laser light according to an easily programmable set of rules and conditions," explains Perrone. "There are all kinds of data that we can collect and use to trigger events. Robotic applications use these lasers to identify physical objects, whether it is vehicles on a roadway or people in an art museum."

Since 2008, Perrone has served as the chief software engineer for LincVolt, an extended-range electric vehicle with advanced telemetry and robotic controls. MAX monitors the performance of critical system components and pushes the information to a server for analysis.

David Clack, a master principal sales consultant on Oracle's Java Sales team, has worked with Perrone for several years. Clack has also worked with Oracle's Java Embedded for ARM and Power Architecture team to create a Java Virtual Machine that is an equal to its mainframe cousin, allowing Java programs to be developed and migrated

Robot hardware platforms from WowWee Robotics that interface with the MAX platform

from the largest multicore CPUs to the smallest microcontrol units.

"Instead of writing custom code for each device, Java allows developers to create universal applets that can be downloaded and updated over the net," says Clack. "Java ties the hardware, the OS, and the development environment together into reusable components. They can run on an embedded controller or a tablet or just about any other computing device."

LIFETIME ACHIEVEMENT

With MAX, Perrone set out to create an environment that fosters portability across hardware, leverages existing tools, and lowers the cost of developing robotic applications. The Java community has eagerly followed his progress, filling the halls at five JavaOne keynote addresses.

Recently Perrone took the helm as chairman of the SAE On-Road Autonomous Vehicle Standards Committee. This position will give him additional opportunities to create new types of Java-based robotic systems and guide the development of this burgeoning industry.

"Anybody can do simple demos," Perrone says. "But to make robots work requires robust systems and a robust rule set. We did our demos in the Grand Challenge and the Urban Challenge. Our next debut is going to be something much more dramatic."

A glimpse of this new technology was revealed in a recent newspaper article describing Perrone Robotics' development of fully autonomous vehicle kits for vehicles with advanced collision avoidance and automated systems that interoperate with fully autonomous soft and collidable vehicles and other targets.

This project, developed with the Insurance Institute for Highway Safety (IIHS), will be used to test the performance and safety of automated vehicles coming onto the market. "This is autonomous vehicle technology being put to work to test, prove the safety of, and ultimately advance the state of the art of autonomous vehicle technology for the benefit of all," concludes Perrone. "Consumers, OEMs, R&D organizations, and government standards and safety organizations alike will benefit from this very tangible and very real technology spearheaded by IIHS, and we're extremely pleased to see it making a directly beneficial societal impact in proving out the safety of semi- and fully automated vehicles." </article>

Based in Santa Barbara, California, **David Baum** writes about innovative businesses, emerging technologies, and compelling lifestyles.

BEN EVANS AND
MARTIJN VERBURG

BIO

PHOTOGRAPHS BY
JOHN BLYTHE AND BOB ADLER

Part 2

Using Java 8 Lambda Expressions

Exploring lambda expressions for the Java programming language and Java Virtual Machine (JVM).

In this article, we will build on the discussion started in Part 1 of this series, "Exploring Lambda Expressions for the Java Language and the JVM." If you are not familiar with the basic syntax of Java 8 lambda expressions or the philosophy that underlies their design, you should read that article first to familiarize yourself with those concepts.

Also, it is important to realize that lambda expressions are still a moving target. They are intended to be feature-complete as of milestone M6 (end of January 2013), but even after that date, there might be minor changes before release. This means that code from earlier builds of lambdas might not always work with more-recent beta builds. In this article, we will describe some major changes that have occurred in the implementation of lambdas since the first article. Welcome to life on the bleeding edge.

To recap, the overall goals of Project Lambda can be summarized as follows:

- Allow developers to write cleaner and more-concise code.
- Provide a modern upgrade to the Java collections libraries.
- Provide better support for multicore processors (including automatic parallelization).

We discussed the writing of more-concise code using lambdas in the first article. In this installment, we discuss the upgrades to the collections libraries and the Stream abstraction. Parallelization is a big subject, so we defer a full discussion of it to a later article.

Backward Compatibility

One of the most important concepts in the Java platform is that of backward compatibility. The guiding philosophy has always been that code that was written or compiled for an earlier version of the platform must continue to work with later releases of the platform. This principle allows developers to have a greater degree of confidence that an upgrade of their Java platform software will not affect currently working applications.

However, as a consequence of backward compatibility, there is a platform limitation that can affect developers: the Java platform may not add additional methods to an existing interface. To see why this is the case, consider the following. If a new version of an interface I were to add a new method newWithPlatformReleaseN() with release N of the Java platform, all previous implementations of I that were compiled with platform version N-1 (or earlier) would be missing this new method. This would cause a failure to link old implementations of I under Java platform version N.

BY DEFAULT
The default-methods mechanism works by modifying class loading.

This limitation is a serious concern for the Java 8 implementation of lambda expressions, because a primary design goal is to be able to use lambda expressions throughout the Java collections libraries. This goal will, in turn, allow the standard Java data structures to implement coding idioms that come from the functional school of programming.

Default Methods

In order to solve the backward compatibility limitation, an entirely new mechanism was needed. The goal was to allow the upgrade, or evolution, of interfaces with new releases of the Java platform. This mechanism is referred to as *default methods*.

From Java 8 onward, a default method (sometimes called an *optional method*) can be added to any interface. It must include an implementation, called the *default implementation*, which is written inline in the interface definition. This represents an evolution of the interface definition and does not break backward compatibility.

The rules governing default methods are as follows:

- Any implementation of the interface *may* (but is *not* required to) implement a default method.
- If an implementing class implements a default method, the implementation in the class is used.
- If an implementing class does *not* implement a default method, the default implementation (from the interface definition) is used.

Let's take a quick look at an example. One of the default methods that has been added to List in Java 8 is the sort() method. Its definition in the List interface is shown in **Listing 1**. This addition means that in Java 8, any List object has an instance method sort() that can be used to sort the list in place using a suitable Comparator.

The default-methods mechanism works by modifying class loading. When an implementation of an interface is being loaded, the class file is examined to see whether all the optional methods are present. If they are, class loading continues normally. If they are not, the bytecode of the implementation is patched to add in the default implementation of the missing methods.

Default methods represent a fundamental change in Java's approach to object orientation. From Java 8 onward, interfaces can contain implementation code. Many developers see this as relaxing some of Java's strict single-inheritance rules.

There is one detail about how default methods work that developers should understand: the possibility of *default implementation clash*. If an implementing class already has a method that has the same name and signature as a new default method, the pre-existing implementation will always be used in preference to the default implementation.

Streams

Recall that one of the goals of Project Lambda was to provide the Java language with the ability to easily express techniques from functional programming. For example, this means that Java will acquire simple ways to write map() and filter() idioms.

Originally, these idioms were implemented by adding methods directly to the classic Java collections interfaces as default methods. However, because "map" and "filter" are relatively common names, it was felt that the risk of default implementation clash was too high—many user-written implementations of the collections would have existing methods that would not respect the intended semantics of the new methods.

Instead, a new abstraction, called Stream, was invented. You can think of a Stream as being analogous to an Iterator for the new approach to collections code. The Stream interface is where all the new "functionally oriented" methods, such as map(), filter(), reduce(), forEach(), into(), and flatMap(), have been placed.

A Stream is best viewed as a consumable sequence of elements that are accessed one at a time (at least for serial streams). This means that after an element has been taken from a Stream, it is no longer available, in much the same way as for an Iterator.

The original collections, such as List and Map, have been given a new default method, stream(). This method returns a stream object for the collection, in a similar fashion to how iterator() was used in older code that uses collections.

Example: Rewriting Old Lambdas Code

The code in **Listing 2** shows how we can use a Stream and a lambda expression to implement a filter idiom in Java 8. This syntax has changed a bit since the first lambda article; we now have to include a stream() call because List no longer has a filter() default method.

The second thing that has changed is that we also need to

LISTING 1 LISTING 2

```
public default void sort(Comparator<? super E> c) {
  Collections.<E>sort(this, c);
}
```

Download all listings in this issue as text

call into() because filter() no longer returns a Collection; instead it returns another Stream. In order to get a Collection back, after our filtering operation, we need to use into() to convert the Stream to a Collection. The overall approach looks like **Listing 3**.

The idea is for the developer to build a "pipeline" of operations that need to be applied to the Stream. The actual content of the operations will be expressed by using a lambda expression for each operation. At the end of the pipeline, the results need to be materialized back into a Collection, so the into() method is used.

Let's look at part of the definition of the Stream interface, which defines the map() and filter() methods (see **Listing 4**). Don't worry about the scary-looking generics in those definitions. All the ? super and ? extends clauses mean is, "Do the right thing in cases where the objects in the stream have subclasses."

These definitions involve two new interfaces: Predicate and Function (which was called Mapper in early implementations of lambdas). These interfaces can both be found in the java.util.function package, which is new for Java 8. Both interfaces have only one method, which doesn't have a default. This means that we can write a lambda expression for them, which will be automatically converted into an instance of the correct type by the Java runtime.

Remember that conversion to the correct "functional interface" type (via type inference) is always what the Java platform does when it encounters a lambda expression. See the first article for details.

Let's look at the code example shown in **Listing 5**. Suppose we're modeling otter populations. Some are wild and some are in wildlife parks. We want to know how many caged otters are looked after by female zookeepers. With lambda expressions and streams, this is easy to do.

INTRODUCING STREAM

A new abstraction, called Stream, was invented, which is analogous to an Iterator. The Stream interface is where all the new "functionally oriented" methods have been placed.

First, we filter the stream so that only captive otters are handled. Then, we use map() to get a stream of keepers, rather than the stream of otters (note that the type of this stream has changed from Stream<Otter> to Stream<Keeper>).

LISTING 3 / LISTING 4 / LISTING 5 / LISTING 6

```
    stream()  filter()  map()  into()
Collection -> Stream -> Stream -> Stream -> Collection
```

Download all listings in this issue as text

Then, we filter again to select only the female keepers, and then we materialize this into a concrete collection instance. Finally, we use the familiar size() method to return the size.

In this example, we have cleanly transformed our otters into the appropriate keepers, and we didn't mutate any state to do so; this is sometimes called being *side-effect free*. In Java, the convention is that code inside map() and filter() expressions should always be side-effect free. However, this is not enforced, so be careful.

Instead, if we want to mutate some external state, we would use one of two approaches. If we want to build up aggregate state (for example, a running total of the ages of the otters), we would use reduce(), whereas for more-general state alteration (for example, transferring otters to a new keeper when the old one leaves), forEach() is more appropriate.

Let's examine how we would calculate the otters' average age using the reduce() method (see **Listing 6**). First, we map from the otters to their ages. Next, we use the reduce() operation, which takes two arguments: the initial value (often called *the zero*) and a function to apply step by step. In our example, the function is just a simple addition, because we want to sum the ages of all the otters. Finally, we divide the total age by

LISTING 7

```
ots.stream()
  .filter(o -> !o.isWild())
  .filter(o -> o.getKeeper().equals(kate))
  .forEach(o -> o.setKeeper(bob));
```

Download all listings in this issue as text

the number of otters we have.

Notice that the second argument to reduce() is a two-argument lambda. The simple way to think about this is that the first of the two arguments is the running total of the aggregate operation.

Finally, let's turn to the general case in which we want to alter state. For this, we will use the forEach() operation. In our example, we want to model the Keeper kate going on holiday. So all her otters should be handed over to bob for now. This is easily accomplished as shown in **Listing 7**.

Notice that neither reduce() nor forEach() uses into(). This is because reduce() accumulates state as it runs over the stream, and forEach() is simply applying an action to everything on the stream. In both cases, there's no need to rematerialize the stream.

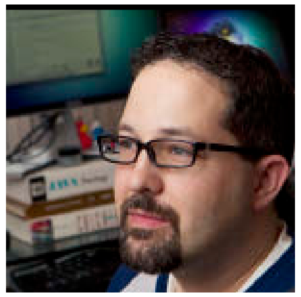
Conclusion

In this article, we've shown how default methods are allowing the Java collections framework to evolve via the new Stream approach. This approach allows functional idioms, such as map(), filter(), reduce(), and forEach(), to be used in version 8 of the Java platform to produce cleaner, more-readable code.

In the next article, we will talk about more-advanced topics, such as lazy and eager evaluation, primitive streams, and automatic parallelization. </article>

LEARN MORE

- For help with lambda expressions or to join our global hack days, join the Adopt OpenJDK project.



JOSH MARINACCI

BIO

Part 1

The Advanced Java Compiler API

Learn how to take advantage of the Java Compiler API.

PHOTOGRAPH BY
CHRIS PIETSCH/GETTY IMAGES

Interest in innovative integrated development environments (IDEs) is on the rise. You might have recently seen the Light Table Kickstarter project, which aims to change how we write code by letting us process and generate code through a unique interface. Such code manipulation tricks require the ability to fully parse, analyze, and manipulate the target language.

Most examples of fancy code manipulation have been performed using JavaScript or Lisp derivatives, which make manipulation easy to do. While this is often cited as one of the powers of dynamic languages, we can, in fact, do these same things with Java, thanks to the Java Tools API (javax.tools), which enables us to programmatically load, parse, analyze, and generate class files from our Java codebase.

In this two-part series, we will explore large chunks of the Java Compiler API. The source code for the examples described in this article can be downloaded here.

Let's Compile Some Code

The Java Compiler API refers to the javax.tools package. In theory, javax.tools will eventually provide access to all the standard Java tools such as javah, javap, and javadoc. Currently, however, we can access only the compiler. To get a reference to a compiler, first we call ToolProvider.getSystemJavaCompiler(). This will return a JavaCompiler object that represents the system compiler. Keep in mind that this will work only if you already have a Java compiler installed. If you have only a Java runtime environment (JRE) instead of a JDK, it won't work. As a software developer, you will, of course, have the JDK installed, but it might be an issue if you want to use the Java Compiler API in an end-user application.

Once we have a compiler, we need to tell it what to compile. This is done with a JavaFileManager. The API can handle many implementations of a file manager, but to keep things simple, we will use the default: StandardJavaFileManager. **Listing 1** shows the code to get a compiler and StandardJavaFileManager.

Notice that I have passed in several nulls to use the default values. The standard JavaFileManager is very customizable, but nulls will give us the defaults. The file manager can turn regular files into special file objects that the compiler understands called JavaFileObjects. We do the conversion with the getJavaFileObjects() call, as shown in **Listing 2**.

FEEL THE POWER

The Java compiler is very powerful. With these APIs we can use this power to do some very interesting and productive things.

Now that we have a file manager, some files, and a compiler object in hand, we can compile something. These three elements are combined to get a CompilationTask (see **Listing 3**), which is the object that will actually do the compiling. Again, notice the extra nulls to accept defaults.

The last line in **Listing 3**, task.call(), is what actually invokes the compiler. Until this call, nothing really happens. task.call() tells the system to actually load up the source files, parse them, look for errors, and then produce the final bytecode as class files. In essence, this is all there is to using the compiler. Everything else we do will expand on this standard core.

Why the API Is Indirect

You might ask why this API has so much indirection. Java already has the java.io.File class, so why do we need the file manager and special JavaFileObject classes? The Java Compiler API was designed to completely isolate the underlying compiler. While we are compiling only a single file from disk, there are many other options. With a custom FileManager, we could have multiple source directories in special places, or we could load source files from the network. We could even generate code in memory and never have it on disk—perhaps storing the code in a remote database. The Java Compiler API was designed to be flexible enough to handle these situations. Unfortunately, this flexibility makes the common case a bit more complicated than it needs to be.

Processing Code with the Annotation Processing API

So far, we have just invoked the compiler on a Java file. We haven't done anything that we couldn't have done from the command line. To do more-interesting things, we need to build a processor that will look at the code as it is compiled.

Java 5 introduced the Annotation Processing API, which was extended in Java 6 and Java 7. This API lets you find annotations in a codebase and work with them. Annotations are often used in Web frameworks such as Hibernate and Spring to replace verbose XML files.

An annotation processor is a class that will be called during compilation as the compiler travels through each package, class, and method. The API was designed to let you work with the parts you care about and ignore the rest. To start, we will create a simple processor that lists every class (see **Listing 4**).

Our processor, called SimpleProcessor, extends the AbstractProcessor class provided by the API and prints a list of processed elements. The process method will be called each time the compiler goes through a round of processing. Some processors might modify the code, so the compiler must go through multiple rounds, invoking each processor on every round. Because we want to do the processing only once, I check for env.processingOver() and do the loop only if we are still in the processing phase. The SimpleProcessor is attached to the compiler task with a task.setProcessors() call, as shown in **Listing 5**. Notice that setProcessors takes a list, not just a single reference. You can attach as many processors as you want to a compilation. This makes it easy to mix your new processor with other processors that you might get from a library. When the compiler is invoked with task.call(), the output shown in **Listing 6** will be printed.

The elements array is actually the root of a tree. There is a tree node for each class. There are child nodes for each field, constructor, and method under each class. With these elements, you can traverse the entire API of your code.

Before we continue, I want to point out two very important lines in **Listing 4**. Above the processor definition are two annotations:

- The SupportedSourceVersion annotation tells the version of Java code this processor is designed to work with. Setting it lets the compiler adapt properly if your processor gets used in a future version of Java in which the syntax might change (for example, when lambda expressions come in Java 8).

LISTING 1 / LISTING 2 / LISTING 3 / LISTING 4 / LISTING 5 / LISTING 6

```
JavaCompiler compiler = ToolProvider.getSystemJavaCompiler();
StandardJavaFileManager fileManager =
 compiler.getStandardFileManager(null, null, null);
```

Download all listings in this issue as text

- The other annotation, SupportedAnnotationTypes, tells which annotations your processor will look for. The Annotation Processing API, as the name suggests, was originally designed to work only with annotated classes.

SupportedAnnotationTypes lets you control which classes will be processed, which is good if you have a large codebase and want to work with only a few classes. However, if we set SupportedAnnotationTypes to *, our processor will be called for all classes, even the ones without annotations. We have called upon the Annotation Processing API to become a general-purpose source code parser, ready to do our bidding.

Doing Something Useful with Processors

Listing 7 is an updated version of SimpleProcessor. The code loops through each class and then each child element of each class. The Element interface has an ElementKind enumeration to let you determine what kind of element the object is. This allows us to separately count the fields from the methods. Notice that I am counting both the METHOD and CONSTRUCTOR kinds as methods.

Fixing Some Bugs

So far I have been using a test class called TestClass.java. **Listing 8** shows the code.

TestClass actually has four methods in it, not three as the processor says. What's wrong? Notice that TestClass has an inner class called TestInnerClass. The extra method is in there. Our processing code only goes one level deep, so it missed the method of the inner class.

There are two ways to fix this. First, we could rewrite our loop to go down another level, or we could rewrite it to be recursive. However, there is a better way. Because traversing through the element tree is such a common task, the Compiler API has utility classes to help us. We will use ElementScanner6.

ElementScanner6 is a concrete class that will loop through every element in the entire tree. It has several methods whose names begin with *visit*. You need to implement only the methods for the types of things that you want to visit. Everything else will be skipped. In our case, we want to visit the classes, methods, and fields, so we will override visitType, visitExecutable, and visitVariable.

ElementScanner6 takes two generic arguments: one for a parameter to be passed to each visit method and one as the return type of each method. This lets you do a map-reduce style of programming when you want to perform parallel analysis of large codebases. Because our example doesn't need either parameter, I have set them both to Void in **Listing 9**.

Notice that in each overridden method, we call the super method as the last line. This ensures that our code will be called on each element before the element's children. This is called a *preorder traversal*. By modifying the position of

LISTING 7 / LISTING 8 / LISTING 9

```
public static class SimpleProcessor extends AbstractProcessor {
  public boolean process(
   Set<? extends TypeElement> types, RoundEnvironment env) {
    if(env.processingOver()) return false;
    int classCount = 0;
    int methodCount = 0;
    int fieldCount = 0;

    for(Element elem : env.getRootElements()) {
      if(elem.getKind() == ElementKind.CLASS)
        classCount++;
      for(Element sub : elem.getEnclosedElements()) {
        if(sub.getKind() == ElementKind.FIELD)
          fieldCount++;
        if(sub.getKind() == ElementKind.METHOD
            || sub.getKind() ==
             ElementKind.CONSTRUCTOR)
          methodCount++;
      }
    }

    u.p("total class count: " + classCount);
    u.p("total method count " + methodCount);
    u.p("total field count  " + fieldCount);
    return false;
  }
}
```

Download all listings in this issue as text

the call to super, we could also perform post- or in-order traversals.

As you learn more about the Java Compiler API, you will start to pick up the specific vocabulary that it uses. For example, we use the visitType method rather than visitClass to find classes. Technically, a type includes more than just classes. It can also include enums and interfaces. *Type* is the catchall word for these things. The term *executable* covers both methods and constructors, but you can distinguish between them by the ElementKind. The term *variable* covers both fields and method arguments. The visitVariable method in **Listing 9** checks to see whether the parent element of the variable is a class. If it is, the variable must be a field. If it isn't, we just ignore it.

If we run the code in **Listing 9**, we will get the following results:

```
total class count: 2
total method count: 5
total field count: 3
```

What? Now we get five methods instead of three, but there are only four methods in the actual test file. What gives? Well, we have to remember that the Java language specifies that each class must have at least one constructor. If you don't create a constructor, the compiler will create one for you. The TestInnerClass class did not have an explicit constructor, so the compiler generated one, which accounts for the extra method.

Processing More than One Class

So far we have processed just one top-level class: TestClass. In the real world, we'd like to process an entire codebase. Compiler.getTask takes an Iterable of files, so we could manually provide a list of every source file to be processed, but that is a lot of work. Instead, we can set a source path that the compiler will recursively follow looking for Java files, just like we can do on the command line. We can set this up using the file manager (see **Listing 10**).

The file manager has a concept of *locations*, which are special places that the compiler will look for things. Source and Output are standard locations that the compiler already knows about, so we just need to set those locations to real file paths. fileManager.setLocation accepts a List of files so you can provide more than one source location if you wish. For our purposes, we will just use a single file pointing to XMLLib/src, which is a small XML processing library that I wrote. I also set the CLASS_OUTPUT location to /tmp rather than leaving it as the default.

LISTING 10

```
StandardJavaFileManager fileManager =
  compiler.getStandardFileManager(null, null, null);
fileManager.setLocation(
  StandardLocation.CLASS_OUTPUT, Arrays.asList(new File("/tmp")));
      File pth = new File(
        "/Users/josh/projects/Leo/LeonardoSketch/Sketch/src/");
fileManager.setLocation(
  StandardLocation.SOURCE_PATH, Arrays.asList(pth));

Set<JavaFileObject.Kind> kinds =
  new HashSet<JavaFileObject.Kind>();
kinds.add(JavaFileObject.Kind.SOURCE);

Iterable<JavaFileObject> files = fileManager.list(
      StandardLocation.SOURCE_PATH, "", kinds, true);
JavaCompiler.CompilationTask task = compiler.getTask(
      null, fileManager, null, null, null, files);
```

Download all listings in this issue as text

Once the source location is set, we still must load up those files using filemanager.list(). The list() method requires a Set of Kind objects to filter the files. I have used a Set containing only the Kind.SOURCE constant so that it will pick up only Java source files. There are also constants for HTML, CLASS, and OTHER. The list method returns the final list of files to give to the compiler task.

Visualizing the Codebase

Now that we have a structure that represents our entire codebase, we can finally do something interesting with it: draw a picture. We will draw a chart of the classes.

The packages are laid out horizontally, with the classes in each package drawn below the packages. Each class has a name and lines drawn to the other classes that it references. This will give us a visualization of all the classes and which ones talk to others the most. These kinds of visualizations help pick up high-level patterns in a large codebase that might otherwise be missed when we are coding. Because this example requires a fair amount of code, I will show only

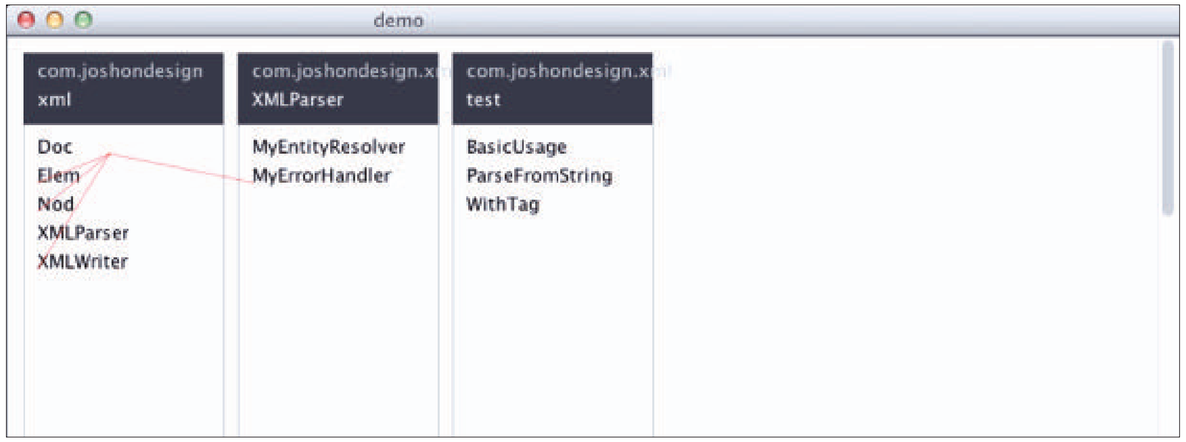


Figure 1

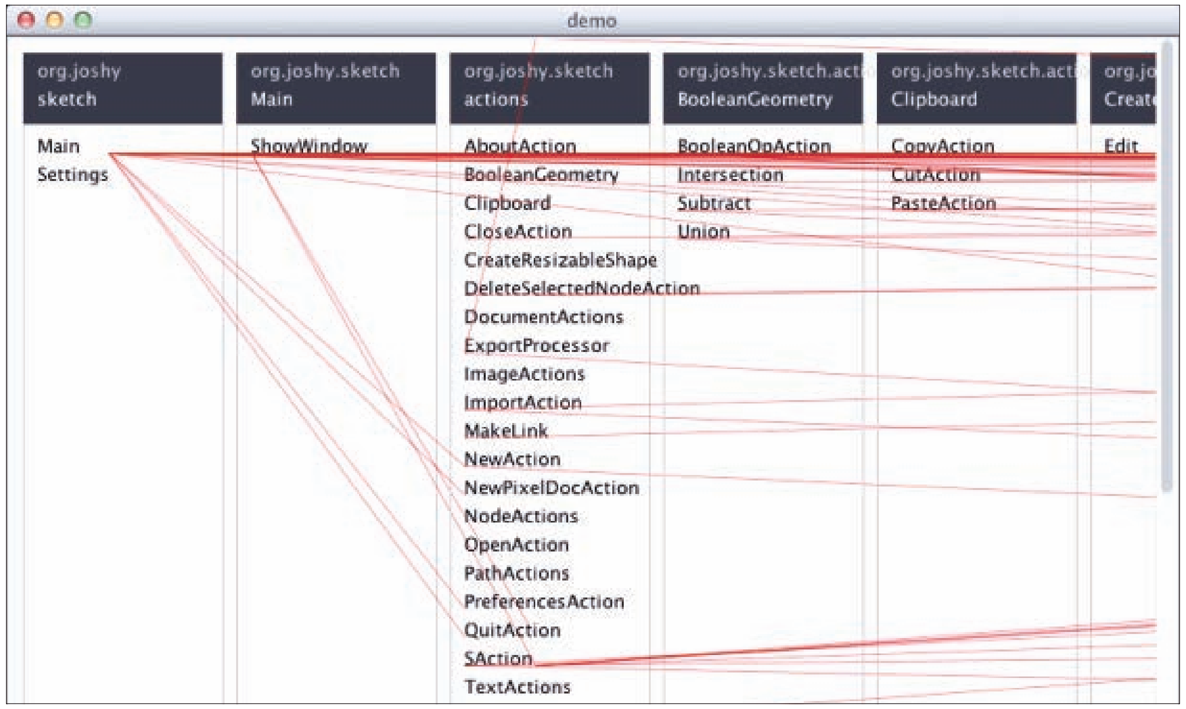


Figure 2

the highlights.

First, I created a data model class called Claz that represents a class definition in the codebase. It stores the name, package, and sorted list of class references for each class (see **Listing 11**).

Next, I created a new processor called CodeCounter. This processor tracks every class it sees by name and package. For each class, it looks for references to other classes, such as a method parameter of those other class types. **Listing 12** shows an abbreviated version of the code of CodeCounter.

Finally, I created a Swing component that will draw the data model called GraphComponent. When first created, it sorts all the class references into alphabetical order and then draws them on the screen as boxes and lines. **Figure 1** shows what the screen looks like after parsing my XMLLib project's codebase.

LISTING 11 LISTING 12

```
private static class Claz implements Comparable<Claz> {
    String qname;
    String sname;
    String pkgname;
    Set<String> refs = new HashSet<String>();
    List<String> sortedRefs = new ArrayList<String>();

    public int compareTo(Claz claz) {
        return this.qname.compareTo(claz.qname);
    }
}
```

Download all listings in this issue as text

Of course, this is a pretty small codebase. To really show off what it can do, I ran it again on the code for Leonardo Sketch, a full-featured vector drawing tool I've been working on that has several hundred classes. **Figure 2** shows what the output looks like.

Notice the size of the horizontal scroll bar. This codebase has a *lot* of classes, yet it took only about four seconds for it to run and draw on my laptop. The Java compiler is pretty fast.

Conclusion

If you study the output carefully, you will notice something. This processor finds references to other classes that are used as method arguments or fields, but it won't find variables that are inside of methods. This reveals something interesting: the Annotation Processing framework, as powerful as it is, will process only the API of your codebase. It will find classes, methods, and fields, but it will not go into the implementation of your methods.

FIND ANNOTATIONS
The Annotation Processing API lets you find and work with annotations, which are often used in Web frameworks such as Hibernate and Spring to replace verbose XML files.

It is *not* actually processing the logic and statements of your code. It is missing the if statements, for loops, and every other statement that makes up the Java language. The underlying compiler does process these things, of course; it just doesn't expose them to us. To really see everything in our code, we will have to go beyond the Annotation Processing API.

In Part 2 of this series, we will dive into the advanced compiler APIs to fully process Java source down to the statement level. We will build a full custom-code browser that lets you browse your entire class structure, report errors, see your current navigation path, and display helpful information as you browse code. To do this, we will begin using the javax.lang.model and com.sun.source APIs.

The Java compiler is very powerful. With these APIs we can use this power to do some very interesting and productive things. </article>

LEARN MORE

- javax.tools package
- Java Compiler API

MAKE THE FUTURE JAVA
Java
FIND YOUR JUG HERE
One of the most elevating things in the world is to build up a community where you can hang out with your geek friends, educate each other, create values, and give experience to you members.
Csaba Toth
Nashville, TN Java Users' Group (NJUG)
LEARN MORE
ORACLE

KITO D. MANN

BIO

Responsive Interportlet Communication with Ajax

Learn how to build portlets that communicate with each other and update themselves dynamically on the client.

Portal servers occupy a unique niche in the enterprise Java world. When you mention them to other Java developers, some cringe, some say they're clueless, and others see lots of value and potential. Originally, the goal of Java-based portal servers was to provide unified Web-based access to information and applications, usually within an organization (intranet), between organizations (extranet), or with external customers (internet). Over the years, they have evolved to provide a wide range of features, including the following:

- User personalization
- Back-end customization
- Single sign-on (SSO)
- Content aggregation
- Navigation
- Content management
- Social networking ("Enterprise 2.0")
- Collaboration tools
- Theming and layout
- Mobile support
- Application API

Portal servers are powerful application platforms that provide many features that developers often have to build themselves. While heavy-duty enterprise-class portal servers, such as Oracle WebCenter and IBM WebSphere Portal, are alive and well, there are also some great open source products, such as GateIn and Liferay. Conceptually, non-Java products such as Joomla and Drupal can count as portals, but they are often called *content management systems*.

Portlets, which are the application building blocks of portals, can provide the same functionality as an ordinary Web application, but they run inside a portal server, which means that they can be arranged on different portal pages and access to them can be controlled by the server. The Portlet API, which is supported by all major portal servers, provides a standard way to build portlets.

A key benefit of using portlets is the ability to write application components with a narrow focus. For example, you could write a general-purpose portlet (such as a Google Maps widget), a line-of-business portlet (such as an application that creates a new insurance claim), or a portlet that provides a particular view of an internal system (for instance, all of the outstanding issues for the current user in the internal issue tracker). The portlet doesn't have to worry about how permissions are defined, the overall look and feel, or its layout on the page; it can focus on the core functionality and leave the rest to the portal.

Portlets are usually built using ordinary Web frameworks such as JavaServer Faces (JSF), Spring MVC, Wicket, and so on. A single Web application archive (WAR) can contain any number of portlets and servlets, which might or might not work together. The actual portlet integration is provided via a *portlet bridge*, which adapts the Web framework to work with the Portlet API, allowing you to continue to use the Web framework's programming model.

Note: All the examples in this article were tested on Liferay Portal and use the Liferay Faces Bridge. Complete versions of the examples, with all other artifacts such as configuration files, are available on GitHub.

Special thanks to Neil Griffin for his excellent examples, which were used as a starting point, and for his support while writing this article.

How Do We Communicate?

Because portlets usually focus on a specific set of features, they can be deployed in a variety of contexts. You might have a video player portlet that plays an arbitrary video from the Web, but if it's on the same page as a content management portlet, it will play the selected video in the content management portlet. In order for this to work, the portlets need to talk to each other.

Portlet API 2.0 (the most recent version) provides three options for interportlet communication (IPC): public render parameters, events, and session state.

Public render parameters. Portlets receive different types of requests, three of which are *render*, *action*, and *event*. Render requests display the portlet's content; action requests perform an application activity requested by the user (such as updating data in the database); and event requests are generated when another portlet raises an event.

When a portlet receives a render request, it can use *render parameters* to help it determine what to display. (Unlike ordinary servlet-based applications, a portlet can be rendered because another portlet has caused a full-page refresh). Usually, render parameters are scoped at the portlet level, so they're not shared among different portlets. However, you can mark specific parameters as public, which means they can be used by other portlets on the page. This way, when an action occurs in your portlet, you can set a public render parameter, and then any other portlets on the page can use it when the page is refreshed. This provides a lightweight, loosely coupled mechanism for IPC.

Public render parameters work for portlets in different Web applications, and because they are request parameters, the value must be a string. In JSF portlets, public render parameters can be automatically mapped to object properties. This way, properties are updated automatically when public render parameters are available, and public render parameters are generated automatically when a property changes.

Usually, public render parameters are scoped to a particular portal page, so if you want to communicate with a portlet on a different page, you're out of luck. Some portal servers have options to remove this restriction, however.

Events. Events allow communication of an event name and an arbitrary payload between different portlets. When a portlet receives an action request, it can generate one or more portlet events. Before the portal server refreshes the page and rerenders all the portlets, receiving portlets can respond to the event.

Like public render parameters, events are decoupled from the Web application, so they allow communication between portlets in different WARs. They're more heavyweight, but they also allow you to send objects instead of just strings. They're also usually scoped to portlets on the same page, but again, portal servers can relax this limitation.

Session state. In ordinary Web applications, different objects (controllers or backing beans, for example) can communicate easily by storing objects in the session. The same holds true for portlet applications. However, the portlet specification defines two different types of session scopes. There's the private session scope, which belongs to a particular portlet (PortletSession.PORTLET_SCOPE), and a shared scope, which is available to all portlets in the Web application (PortletSession.APPLICATION_SCOPE).

How this is handled depends on the *portlet bridge*. The JSF portlet bridge exposes this scope with an implicit variable available via the expression language (EL): #{httpSessionScope}. You can also access the PortletSession object directly. However, you can't use JSF annotations or XML configuration to place an object in this scope.

Using session scope for communication is easy for most developers to understand, and it allows you to store objects easily. Portlets can also talk to each other even if they're on different pages. The drawback, of course, is that all the portlets have to be in the same Web application. This is fine if you're writing several portlets at the same time that were intended to work together. It won't work in situations where the portlet with which you'd like to communicate is in a different Web application, which might be the case if it's owned by

A PORTLET PLUS

A key benefit of using portlets is the ability to write application components with a narrow focus.

another department or is from a third party.

It's worth noting that some portal servers have nonstandard settings that make communication through the session more flexible. Liferay, for example, will share application-scoped session data with other portlet applications if you set a couple of parameters in its configuration file.

Enter Ajax

You might have noticed that I mentioned *full-page refresh* earlier. This is because the Portlet API is a little behind when it comes to building today's rich, interactive applications. The standard portlet lifecycle is pretty old-school: a portlet submits an action request, and the entire portal page (which can include several other portlets) is refreshed. Usually, the other portlets have a chance to change their output (perhaps based on session state, public render parameters, or events) before the page is displayed, but at the end of the day, we're still talking about a full-page refresh. (As usual, some portal servers, such as GateIn, have proprietary ways around this.)

Ajax support is usually achieved through a *resource request*, which can return a response outside of the normal portal lifecycle. This is handy for retrieving things such as images, JavaScript files, and stylesheets and also for returning data (JSON, XML, and so on) or markup. A portlet can send a resource request from the browser via the XMLHttpRequest object and then update itself dynamically based on the response, and the portal doesn't have to rerender other portlets on the page.

The problem is that resource requests are quite limited: they can't generate portlet events or set public render parameters. So if you initiate an Ajax request from a portlet, the only way to communicate directly with another portlet on the server is via the session. And even then, there is no mechanism to automatically update the other portlet's UI. The changes won't be reflected until the entire portal page is refreshed.

There are two ways we can avoid this limitation: Ajax Push and IPC inside the browser.

Pushing Ajax

In order for one portlet to update other portlets from the server, we need a way to push changes back to the browser and avoid a full-page refresh. The process of pushing data to the browser from the server is called *Ajax Push* or *Comet* (other terms are *Reverse Ajax*, *HTTP Streaming*, and *HTTP Server Push*).

The Portlet API doesn't have

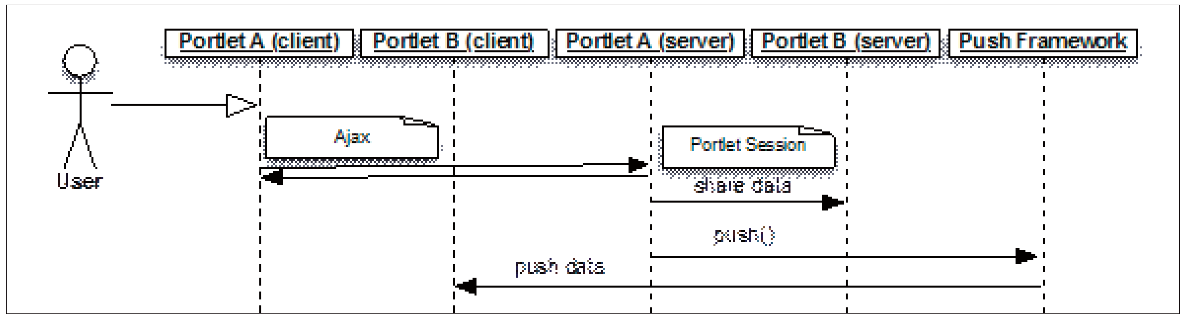


Figure 1

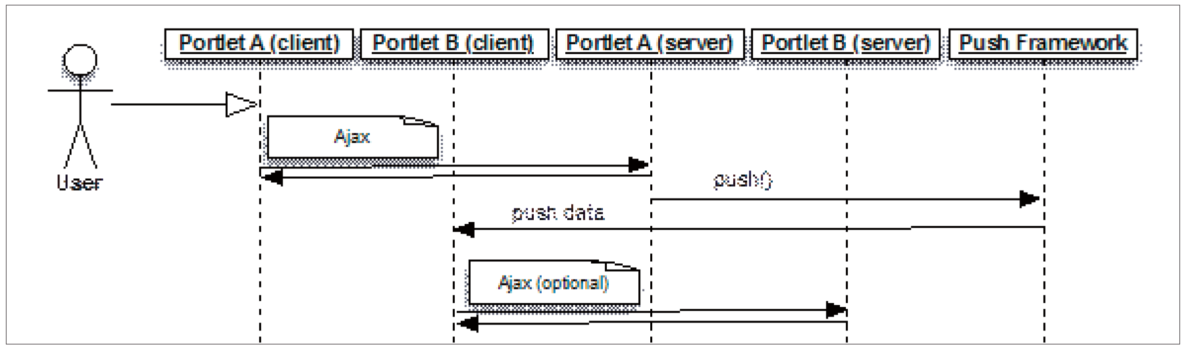


Figure 2

Ajax Push built in, but Ajax Push can be accomplished with third-party libraries. Some popular open source libraries are Atmosphere, CometD, ICEpush, and Direct Web Remoting (DWR). Most Web application frameworks integrate with one of these libraries (for example, there are Atmosphere plug-ins for Grails, Spring, GWT, Wicket, and Vaadin).

In JSF applications, the component libraries provide Ajax Push support, usually by integrating one of these lower-level libraries. For example, ICEfaces uses the internals of ICEpush, while both RichFaces and PrimeFaces use Atmosphere. Oracle ADF Faces Rich Client uses Active Data Services, which is part of the Oracle Application Development Framework stack.

Regardless of which Web framework or Ajax Push library you're using, there are a couple of different patterns. If the portlets are in the same Web application, they can communicate via the portlet session and then update the UI via Ajax Push. The user interacts with Portlet A, Portlet A sends an Ajax request to the server where data is shared with Portlet B, and then updates are pushed to Portlet B, as shown in **Figure 1**.

If the portlets are not in the same Web application, they can't communicate via the portlet session, but data can be pushed from

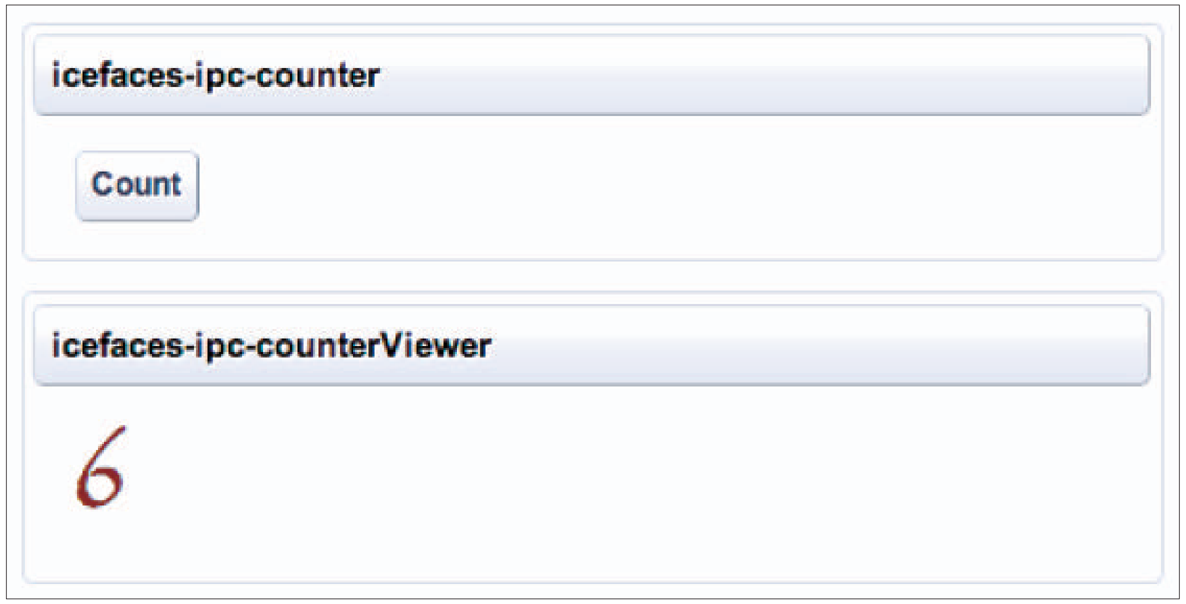


Figure 3

LISTING 1

```
<?xml version="1.0" encoding="UTF-8"?>
<f:view xmlns="http://www.w3.org/1999/xhtml"
  xmlns:aui="http://liferay.com/faces/aui"
  xmlns:f="http://java.sun.com/jsf/core"
  xmlns:h="http://java.sun.com/jsf/html"
  xmlns:ice="http://www.icesoft.com/icefaces/component"
  xmlns:ui="http://java.sun.com/jsf/facelets">
  <h:head>
    <h:outputStylesheet library="example" name="portlet.css" />
  </h:head>
  <h:body>
    <ice:form id="form">
      <ice:commandButton id="count" actionListener="#{counter.incre-
ment}"
        value="Count">
      </ice:commandButton>
    </ice:form>
  </h:body>
</f:view>
```

Download all listings in this issue as text

one portlet on the server to the other portlet on the client. In this scenario, the user interacts with Portlet A, Portlet A sends an Ajax request to the server, and changes are pushed to Portlet B on the client, where it can optionally update its server counterpart via Ajax, as shown in **Figure 2**.

Let's take a closer look at how this works with specific frameworks. We'll look at two possible implementations of a very simple scenario: a Counter portlet that will increase its value by one each time the user clicks a button and a CounterViewer portlet that will dynamically update with the current value of the counter. **Figure 3** shows one version of the two portlets running in Liferay Portal.

ICEfaces and ICEpush. ICEsoft's ICEpush product provides solid Ajax Push capabilities, and it can work with just about any Web framework, such as Spring MVC, Grails, Wicket, and JSF. It uses *long-polling*, which means that a connection with the server is held open in order to provide push notifications. While long-polling is not necessarily the most efficient way to do Ajax Push, it works in all browsers and with all application servers. ICEpush can take advantage of asynchronous request processing (ARP) features in Servlet 3, and ICEsoft has a commercial product available that provides greater scalability.

The ICEfaces JSF component suite uses ICEpush and has a feature called Direct-to-DOM rendering, which automatically updates the relevant portions of the page. What's nice about the approach is that there's nothing special you need to do on the client; you just write a normal JSF application.

Listing 1 shows the Facelet page for the Counter portlet.

This page has a single button that executes the Counter .increment() method when it is clicked by a user. **Listing 2** shows the Counter class, which is a JSF backing bean.

Counter is a request-scoped bean, but the value is stored in the portlet session so that it can be shared with the CounterViewer, which is also request-scoped. In order to access the shared application-wide session instead of the private session, we inject the implicit httpSessionScope variable into the portletSession property.

In order to implement Ajax Push in ICEfaces, you simply register the backing beans in *render groups* using the PushRenderer. When Counter is created, it registers the session with the renderer group named Counter.COUNTER_ RENDER_GROUP. When the bean is destroyed, it will remove the session from the group in the preDestroy() method, which will be called automatically by JSF (thanks to the @PreDestroy annotation).

Ajax Push updates are initiated by calling PushRenderer.render() and passing in the name of the renderer group, as shown in the increment() method. ICEfaces will automatically push updates to any pages in the same group whose state has changed. (Instead of associating the entire session with a render group, you can also associate individual pages.)

The CounterViewer portlet is even simpler than the Counter portlet. The Facelet page, shown in **Listing 3**, has a single component that displays the count from the CounterViewer backing bean, which is shown in **Listing 4**.

CounterViewer simply retrieves the count value from the session. In order to receive push updates, it is registered with the same render group.

That's all there is to it—ICEfaces automatically takes care of updating the CounterViewer UI for us whenever someone clicks on the button in the Counter portlet. For the most part, no additional configuration is required, although there are several options that can be used to affect the push behavior. There is one caveat, however: in ICEFaces Community Edition, this approach works only if all your portlets are in the same Web application (the Enterprise Edition does not have this restriction.)

Atmosphere and PrimeFaces. Atmosphere is essentially the One Push Library to rule them all. Instead of relying on a single transport mechanism such as ICEpush, it picks the most suitable one (long-polling, HTML5 WebSockets, Server-Sent events, HTTP Streaming, or JSON-P) based on the capabilities of the browser and the application server. This means that Atmosphere applications will typically use the most efficient transport available. In order for this to work properly, however, you have to ensure that your application server is properly configured, and in some cases there are bugs or nuances with browser support for features such as WebSockets (make sure you read the wiki).

Atmosphere can work in a variety of environments, including pure JAX-RS, and with different Web frameworks, such as GWT, Grails, Vaadin, Wicket, and JSF via RichFaces or PrimeFaces. The framework is quite capable and has a raft of different features on both the client and the server. However, if Atmosphere is integrated with your Web framework, you can count on an API that eas-

LISTING 2 / LISTING 3 / LISTING 4

```
@RequestScoped
@ManagedBean
public class Counter {

  public static final String COUNTER_KEY = "count";
  public static final String COUNTER_RENDER_GROUP =
"counter-render-group";

  @ManagedProperty("#{httpSessionScope}")
  private Map appSession;

  public Counter() {
    PushRenderer.addCurrentSession(COUNTER_RENDER_GROUP);
  }
  @PreDestroy
  public void preDestroy() {
    PushRenderer.removeCurrentSession(COUNTER_RENDER_GROUP);
  }
  public Integer getValue() {
    Integer value = (Integer) getAppSession().get(Counter.COUNTER_KEY);
    return value == null ? 0 : value;
  }
  private void setValue(Integer value) {
    getAppSession().put(Counter.COUNTER_KEY, value);
  }
  public void increment() {
    setValue(getValue() + 1);
    PushRenderer.render(COUNTER_RENDER_GROUP);
    System.out.println(
"Counter value incremented to: " + getValue());
  }
  public Map getAppSession() {
    return appSession;
  }
  public void setAppSession(Map portletSession) {
    this.appSession = portletSession;
  }
}
```

Download all listings in this issue as text

ily integrates with its programming model. An example of this is the integration with PrimeFaces, which was architected by the Atmosphere lead, Jeanfrancois Arcand, and has a simple-to-use JSF-friendly API.

Let's see how we can build the Counter and CounterViewer portlets using PrimeFaces Push. **Listing 5** shows the Facelet page for the Counter portlet.

You might have noticed that this page is almost exactly the same as **Listing 1**: there is a single button that executes the Counter .increment(). Just like ICEfaces, the push API is accessed in the Counter backing bean, which is shown in **Listing 6**.

Atmosphere sends push notifications to named channels, which are opened from the client. The increment() method increments the Counter value, but it also grabs the PushContext and pushes the new value out to the channel. Note that we append the portlet session ID to the channel name; this ensures that push notifications are received only by portlets in the same session. If we didn't do this, *every* CounterViewer portlet instance on the same server would receive the same updates, which is useful in some cases (such as updating a stock ticker). But here, we're interested in communication between two portlets that are being accessed by the same user.

Now that we've established a channel, the CounterViewer's page needs to listen to it in order to receive updates. PrimeFaces has a handy <p:socket> that encapsulates Atmosphere's JavaScript API in order to handle this for us. We can even use the PrimeFaces <p:ajax> behavior to update the CounterViewer portlet instance on the server when a message is received on the client. **Listing 7** shows the markup for using <p:socket> with <p:ajax>.

The channel attribute grabs the channel name from the CounterViewer.channel property, while the onMessage attribute specifies a JavaScript function called handleMessage(), which responds to notifications received from the server. The <p:ajax> tag creates a behavior that gets fired when a push message is received. This behavior will update the value of the hiddenCountValue element on the server. (Note that it isn't always necessary to update the server after receiving a push message).

Unfortunately, the <p:socket> component assumes the channel is hosted in the same portlet application that generated the page. So in the code snippet in **Listing 7**, the channel would be

LISTING 5 LISTING 6 LISTING 7

```
<?xml version="1.0" encoding="UTF-8"?>
<f:view xmlns="http://www.w3.org/1999/xhtml"
  xmlns:aui="http://liferay.com/faces/aui"
  xmlns:f="http://java.sun.com/jsf/core"
  xmlns:h="http://java.sun.com/jsf/html"
  xmlns:p="http://primefaces.org/ui"
  xmlns:ui="http://java.sun.com/jsf/facelets">
  <h:head>
    <h:outputStylesheet library="example" name="portlet.css" />
  </h:head>
  <h:body>
    <h:form id="form">
      <p:commandButton value="Count" actionListener="#{
counter.increment}" />
    </h:form>
  </h:body>
</f:view>
```

Download all listings in this issue as text

opened with the Web application for CounterViewer. Because the Counter and CounterViewer are in separate portlet applications, and the push request is coming from the Counter application, this won't work for us.

We can, however, work directly with the PrimeFaces JavaScript API in order to gain more control over the URL that is used to open the channel. The code in **Listing 8** is the same as the code generated by <p:socket> and <p:ajax>, but it has a customized URL for the push channel.

Note that the url parameter sent to <p:socket> points to the Counter portlet's application (primepush is the name of the PrimeFaces Push servlet). The expression #{view.getClientId(facesContext)} retrieves the ID of the enclosing element for this portlet (the <p:ajax> behavior automatically adds this prefix).

Listings 9a and **9b** show the entire Facelet page for the CounterViewer.

This page is a little more complicated than the ICEfaces example (**Listing 3**). In addition to providing an output control to display the value property of CounterViewer, we must establish the connection with the server and manually handle notifications. Also, because we're working directly with the PrimeFaces JavaScript API, we must include the primefaces.js and push.js resources manually.

CRINGE WORTHY?
When you mention portal servers to Java developers, some cringe, some say they're clueless, and others see lots of value and potential.

The handleMessage() function receives the payload from the server, which in this case is just the count value from the Counter portlet. It updates an output control in order to display the new value, and it also updates a hidden field with the value. As we discussed earlier, this hidden field is used by the Ajax behavior to send the value back to the CounterViewer backing bean.

The CounterViewer class, shown in **Listing 10**, simply has properties for the counter value and the channel.

As you can see, using Ajax Push with PrimeFaces and Atmosphere is pretty simple from a programming perspective. There is a little bit of extra configuration (you must configure the PrimeFaces Push servlet in web.xml) and, of course, you can tweak the Atmosphere settings to your heart's content. Depending on your chosen application server and browser requirements, you might *have* to do some tweaking in order to get everything up and running (again, read the wiki).

Unlike ICEfaces, PrimeFaces Push doesn't perform automatic UI updates. However, there is no

LISTING 8 / LISTING 9a / LISTING 9b / LISTING 10

```
channelViewerSocket = new PrimeFaces.widget.Socket({
  url:"/primefaces-ipc-counter/primepush#{counterViewer.
channel}",
  autoConnect:true,
  transport:"websocket",
  fallbackTransport:"long-polling",
  onMessage:handleMessage,
  behaviors: {
    message: function(a) {
    PrimeFaces.ab({
      source:"#{view.getClientId(facesContext)}:form",
      event:"message",
      process:"#{view.getClientId(facesContext)}:form:
hiddenCountValue"
    },
    arguments[1])}
  }
});
```

Download all listings in this issue as text

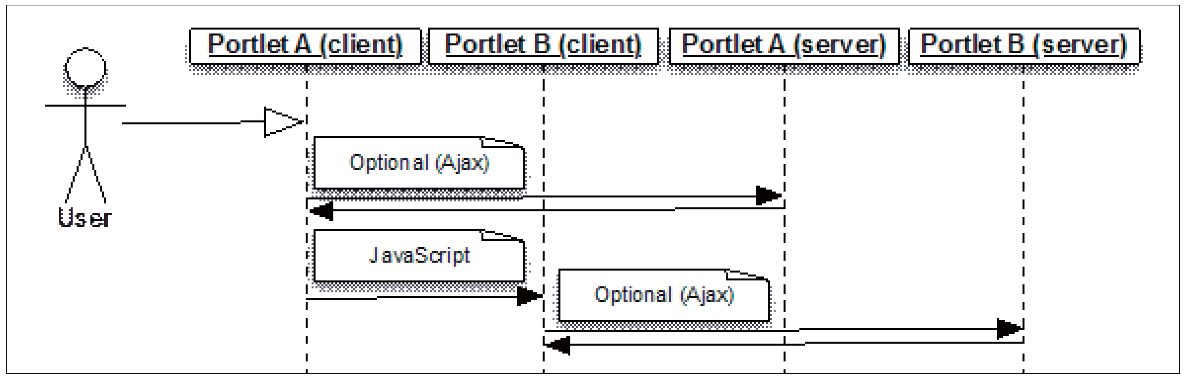


Figure 4

requirement that both portlets be in the same Web application.

Communicating Inside the Browser

An alternative to Ajax Push is to handle IPC in the browser. Portlet A can talk to Portlet B via JavaScript, and then Portlet B can update itself. Either portlet can communicate via Ajax to keep the server in the loop. This approach is shown in **Figure 4**.

If you are building a few custom portlets, it might be tempting to just hardcode the interaction between the portlets in JavaScript. This doesn't scale, however. Portal servers control the Document Object Model (DOM) hierarchy of a given page, and it's difficult, if not impossible, to guarantee that the IDs of DOM elements in your portlet will always have the same identifier. Your code might break when a portlet is redeployed, when the portal server is upgraded to a new version, or when you deploy the portlets on a different portal server.

The trick is to handle communication via a decoupled client-side event bus, which is what the Portlet API provides on the server. Because the Portlet API doesn't have this feature, you can rely on proprietary portal features (Liferay, for example, has a JavaScript API), use the event mechanisms in an existing JavaScript library such as jQuery, or roll your own.

Because jQuery is so pervasive, easy to use, and portable across portal servers, let's see how our Counter and CounterViewer portlets can communicate with jQuery events. **Listing 11** shows the Facelet page for the Counter portlet using jQuery events with standard JSF.

When the user clicks the Count button, we issue an Ajax request using the standard JSF tags, sending in the function sendCount() to be executed for Ajax events (the supported events are error, success, and complete). The Ajax request will execute the

LISTING 11

```
<?xml version="1.0" encoding="UTF-8"?>
<f:view xmlns="http://www.w3.org/1999/xhtml"
  xmlns:aui="http://liferay.com/faces/aui"
  xmlns:f="http://java.sun.com/jsf/core"
  xmlns:h="http://java.sun.com/jsf/html"
  xmlns:ui="http://java.sun.com/jsf/facelets">
  <h:head>
    <h:outputStylesheet library="example" name="portlet.css" />
    <h:outputScript library="example"
name="jquery-1.8.2.js" target="head"/>
  </h:head>
  <h:body>
    <h:outputScript>
    function sendCount(data) {
      if (data.status == "success") {
        var value = document.getElementById(
"#{view.getClientId(facesContext)}:form:
hiddenCountValue")
.value;
        jQuery(document).trigger("count", value);
      }
    }
    </h:outputScript>
    <h:form id="form">
      <h:commandButton id="count"
actionListener="#{counter.increment}"
        value="Count">
        <f:ajax render="hiddenCountValue"
onevent="sendCount" />
      </h:commandButton>
      <h:inputHidden id="hiddenCountValue"
value="#{counter.value}" />
    </h:form>
  </h:body>
</f:view>
```

Download all listings in this issue as text

increment() method on the Counter backing bean on the server (see **Listing 12**) and update the hidden field on the client.

Upon success, sendCount() will first retrieve the value of the counter, which has been returned from the server in a hidden field. Next, it triggers a jQuery event called "count" in the browser, sending in this value. (It's worthwhile to note that some JSF component libraries, such as PrimeFaces, have APIs that allow you to retrieve a payload directly from the Ajax request instead of requiring a hidden field.) Because jQuery events must be attached to a DOM element, we choose the root node— the document—so that the event will always be fired regardless of the page contents.

The CounterViewer portlet simply has to register a function to listen for the "count" event using the jQuery on() method. In order to ensure that the listener works regardless of how the portal server handles JavaScript resources, this work is done using jQuery's ready() method, which executes once the page has been loaded. The CounterViewer Facelet page is shown in **Listing 13**.

Inside of the listener for the "count" event, the portlet retrieves the count value from the Counter portlet, and it then updates one DOM element for display and an invisible input control to send the data back to the server. Next, it uses the JSF Ajax API to send the value of the input control back to the server to update the portlet's backing bean.

Client-side IPC works well in situations where you have control over all of the involved portlets. Depending on your requirements, this approach will require an Ajax request to update the sending portlet, the receiving portlet, or both (as shown in this example). This approach makes more sense for teams that have some JavaScript expertise.

What About the Portlet API?

Resource requests, which are the only portlet requests that can be executed outside the full portal page lifecycle, can't create portlet events or shared public parameters, which are the two built-in IPC mechanisms in the specification. In other words, portlets are forbidden from communicating with each other when responding to Ajax requests. Communication is possible through the portlet session, but even then, there is no built-in way for another portlet to update its UI.

In terms of the Portlet API, two things need to happen in order to have responsive IPC: (1) resource

LISTING 12 LISTING 13

```
@SessionScoped
@ManagedBean
public class Counter {
  private int value;

  public int getValue() {
    return value;
  }
  private void setValue(int value) {
    this.value = value;
  }
  public void increment() {
    setValue(getValue() + 1);
    System.out.println(
"Counter value incremented to: " + getValue());
  }
}
```

Download all listings in this issue as text

requests should be able to generate events and shared public parameters, and (2) portal servers should support partial page refresh (in other words, updating only portlets that have changed).

For (2), the ideal solution would be to support normal Ajax requests and responses with a customized Ajax lifecycle. This is what we've done with JSF: for Ajax requests, the framework executes specific UI components. The response contains only the necessary components, and JavaScript code in the browser updates the appropriate parts of the DOM. Instead of performing a full-page refresh, the portal server could send back specific updates from portlets whose UI has changed and provide a JavaScript callback for each portlet to apply those changes.

Fortunately, we are now beginning to discuss the next version of the Portlet specification. Improving the flexibility of Ajax (and, consequently, resource requests) and supporting WebSockets via the up-and-coming JSR 356 (Java API for WebSocket) are on the list of possible enhancements.

Conclusion

In this article, we discussed several different ways for two or more portlets to communicate. The portlet specification has a few built-in options: public render parameters, portlet events, and the portlet application session. Unfortunately, these options don't natively support partial page updates that users typically expect in today's Ajax-based applications. This can, however, be achieved by using Ajax Push.

In the article, we covered examples using ICEfaces (and ICEpush) as well as PrimeFaces (and Atmosphere). As an alternative, partial page updates can be handled on the client side, so we examined how to accomplish this using jQuery events. (Keep in mind that these techniques might not work for every portal server.)

The next version of the Portlet specification will most likely support partial page updates natively, but for now, you can build responsive portlets that communicate via Ajax Push or JavaScript libraries such as jQuery. </article>

LEARN MORE

- Liferay Faces Bridge
- JBoss Portlet Bridge
- Portlet Specification 2.0
- JSFCentral.com
- Kito Mann's Java.net blog
- Kito Mann's main blog

MAKE THE FUTURE JAVA
Java™
FIND YOUR JUG HERE
My local and global JUGs are great places to network both for knowledge and work. My global JUG introduces me to Java developers all over the world.
Régina ten Bruggencate
JDuchess
LEARN MORE
ORACLE®

STEPHEN CHIN

BIO

Part 2

JavaFX in Spring

Take advantage of Spring to build out the core data screens of your JavaFX application.

In Part 1 of this series, we showed how you can take advantage of the Spring framework in your JavaFX applications to wire up your user interface via dependency injection (DI). This was introduced with an example customer data application that contained several screens and was backed by a Web service back end.

Here, in Part 2, we show how to build out the core data screens of the application. The UI will be created with a JavaFX TableView that is updated via one-line Web service calls using Spring RestTemplate. In addition, to secure the application, we will add a login page and permissions-based security using Spring authentication and authorization.

At the end of the article, we will show an alternative implementation using Java EE standards–based DI and discuss the pros and cons of using the Spring framework and other Java EE technologies in your client application.

Using the JavaFX TableView

The TableView is one of the most important new controls introduced in JavaFX 2, and it is central to any data-based application, such as the Customer Data Application we are building.

Unlike an HTML table, the TableView is not intended for layout, but instead it is closer to a spreadsheet or the Swing JTable control. It allows you to define the properties of an arbitrary number of columns, which will display data that comes from the properties of the table data objects.

The underlying model is an ObservableList of data objects, which can be directly modified to cause the data rendered in the TableView to be updated dynamically.

Figure 1 visually demonstrates the key elements of a TableView.

To create a table, you don't actually have to worry about instances of TableRow; they are automatically generated based on your ObservableList of items. Simply set up instances of TableColumn for each of your object properties, and optionally create a TableCell factory to customize the nodes created for each cell.

There are two styles of creating objects in JavaFX: the first is standard constructors and setters, and the second is object builders. We will be using the latter in this example to set up the table, because it leads to shorter, more readable code. The method to create the data table is shown in **Listing 1**.

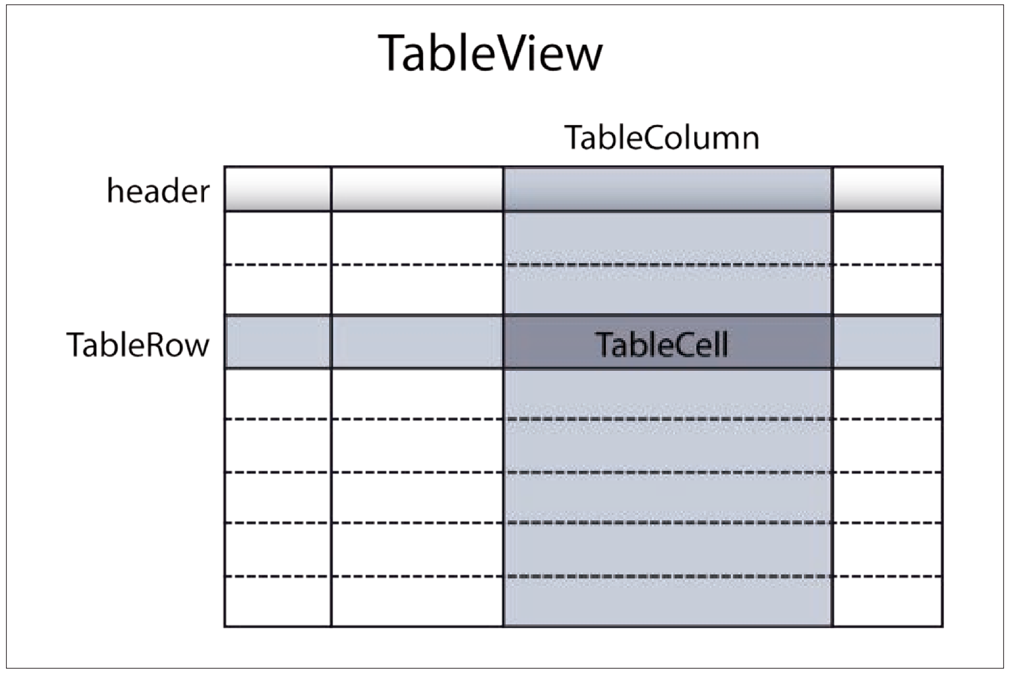
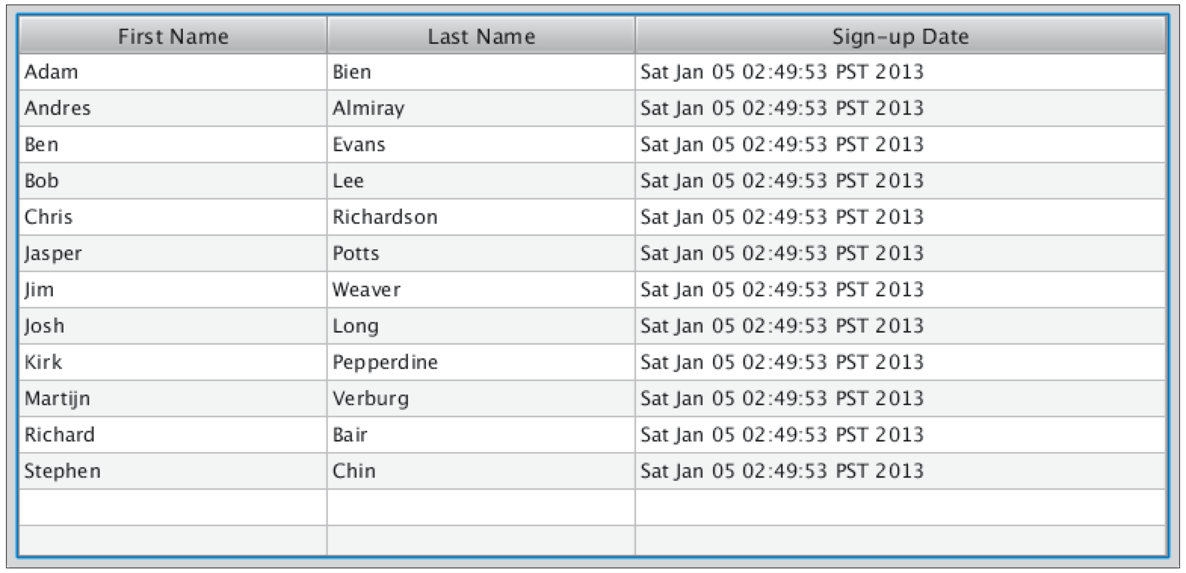


Figure 1



First Name	Last Name	Sign-up Date
Adam	Bien	Sat Jan 05 02:49:53 PST 2013
Andres	Almiray	Sat Jan 05 02:49:53 PST 2013
Ben	Evans	Sat Jan 05 02:49:53 PST 2013
Bob	Lee	Sat Jan 05 02:49:53 PST 2013
Chris	Richardson	Sat Jan 05 02:49:53 PST 2013
Jasper	Potts	Sat Jan 05 02:49:53 PST 2013
Jim	Weaver	Sat Jan 05 02:49:53 PST 2013
Josh	Long	Sat Jan 05 02:49:53 PST 2013
Kirk	Pepperdine	Sat Jan 05 02:49:53 PST 2013
Martijn	Verburg	Sat Jan 05 02:49:53 PST 2013
Richard	Bair	Sat Jan 05 02:49:53 PST 2013
Stephen	Chin	Sat Jan 05 02:49:53 PST 2013

Figure 2

There are a few things to note about the code in **Listing 1**:

- JavaFX builders are fully type-safe. This adds some complexity when declaring generics, but it lets the compiler flag mistakes before runtime.
- Hooking up the model is as simple as providing an ObservableList of items.
- Properties are referenced by name, and the default CellFactory renders them into a text string contained in a Label control.
- To provide some styling for the table, we use inline styles. You can also put this in a CSS file to abstract it from the code.

This code results in a fairly nice-looking data table without a lot of work, as shown in **Figure 2**. (How many of the Java hackers do you recognize in **Figure 2**?) By default, you can resize, sort, and even reorder the columns using drag and drop.

One-Line Web Services

Spring RestTemplate is a great client-side library for accessing HTTP-based Web services. It works transparently with a Spring model-view-controller (MVC) back end, and it can also be used with a standard JAX-RS Web service, given the proper stub objects.

With a little bit of setup, most RestTemplate operations can be accomplished in a single line of Java code. For example, to retrieve a list of all the customers from the back end, all you need to do is call the one-line code snippet shown in **Listing 2**. This snippet supplies the minimalistic parameters: the Web service URL and a compatible model object for the return type.

There is a little bit of setup needed to make this one-liner work. The first requirement is an instance of a RestTemplate, which you can have injected right into your model class using a Spring variable, as shown in **Listing 3**.

LISTING 1 LISTING 2 LISTING 3

```
@SuppressWarnings("unchecked")
private Node createDataTable() {
  StackPane dataTableBorder = new StackPane();
  dataTableBorder.getChildren().add(tableView);
  dataTableBorder.setPadding(new Insets(8));
  dataTableBorder.setStyle("-fx-background-color: lightgray");
  tableView.setItems(controller.getCustomers());
  tableView.getColumns().setAll(
    TableColumnBuilder.<Customer, String>create()
      .text("First Name")
      .cellValueFactory(
new PropertyValueFactory<Customer, String>("firstName"))
      .prefWidth(204)
      .build(),
    TableColumnBuilder.<Customer, String>create()
      .text("Last Name")
      .cellValueFactory(
new PropertyValueFactory<Customer, String>("lastName"))
      .prefWidth(204)
      .build(),
    TableColumnBuilder.<Customer, String>create()
      .text("Sign-up Date")
      .cellValueFactory(
new PropertyValueFactory<Customer, String>("signupDate"))
      .prefWidth(351)
      .build()
  );
  tableView.setPrefHeight(500);
  return dataTableBorder;
}
```

Download all listings in this issue as text

After the variable is configured, you can inject the value using a Bean definition in your CustomerAppConfiguration, as shown in **Listing 4**.

The Customer object is a single class with getters and setters for ID, date, and name. However, it is important to note that this is exactly the same class as that used on the server, so there is zero code redundancy.

Note: For the full source code of all the files, including the Customer object and back-end server code, check out the GitHub project.

Once you have made the RestTemplate call to retrieve the server objects, you can populate your table simply by updating the contents of the table's ObservableList:

```
this.customers.setAll
customers)
```

The code of the model class is shown in **Listing 5** with additional methods for adding and removing customers from the table and back-end Web services.

Notice that adding new customers or deleting customers using RestTemplate is just as easy as retrieving a list and can be accomplished in one line. Similarly, updating the table model is as simple as adding or removing elements from the customers ObservableList and immediately updates the UI.

Authorization and Authentication with Spring Security

Now that we have routines to manipulate data on the server, we need to add a login screen and some client authorization so users cannot erase our database without authorized access. This is an important part of securing your application, but it is not a substitute for having the same security controls on the server side. With proper security on the server and client side, you can get the benefits of instant UI feedback on allowable operations while still protecting your data from hackers.

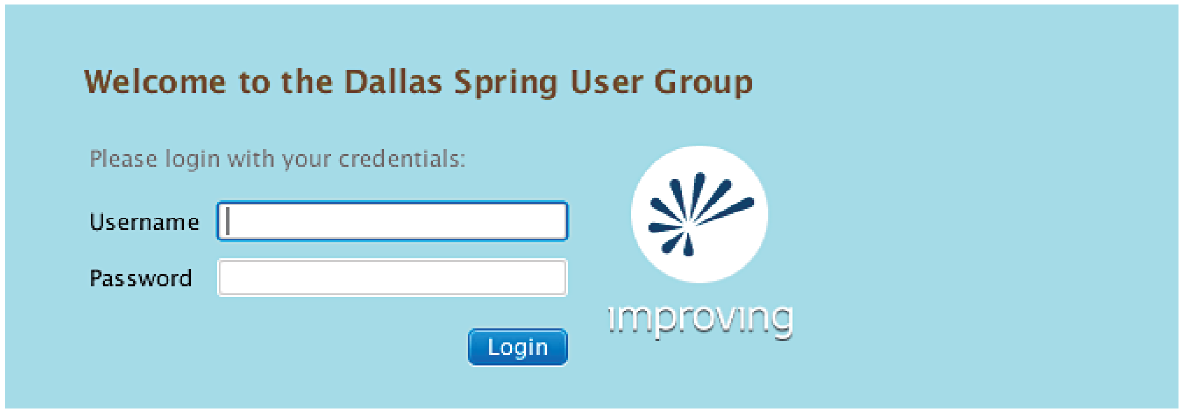
In this article, we show how to create client-side authorization and authentication. Because we are using Spring Security as the underlying model, you will get the benefit of using exactly the same security controls on your server-side application.

The login dialog box is created entirely in FXML using the graphical Scene Builder tool and instantiated with a one-line Bean definition method in our screen configuration file, as shown in **Listing 6**. This will instantiate the FXML login definition and wire it

LISTING 4 LISTING 5 LISTING 6

```
@Bean
RestTemplate restTemplate() {
  RestTemplate restTemplate = new RestTemplate();
  restTemplate.setMessageConverters(
Collections.<HttpMessageConverter<?>>singletonList(
new MappingJacksonHttpMessageConverter()));
  return restTemplate;
}
```

Download all listings in this issue as text


Welcome to the Dallas Spring User Group

Please login with your credentials:

Username

Password

Login

Figure 3

in to our application, giving us the dialog box shown in **Figure 3**.

To enable functionality in the login dialog box, we also need to create a controller class that will provide the login functionality to check credentials using a Spring AuthenticationManager, as shown in **Listing 7**.

The LoginController class makes use of a couple of annotations in different contexts to bind it to the UI dialog box and the Spring AuthenticationManager:

- @Autowired is a Spring annotation that injects a bean instance based on the name or type of the variable. In this case, we are getting access to the AuthenticationManager.
- @FXML, when used on a variable, is a JavaFX annotation that will inject the UI scene graph node with the corresponding fx:id property, so you can manipulate the UI from your controller.
- @FXML, when used on a method, exposes the method to the UI definition so that it can be called on any UI action or event using the #methodName syntax.

The bulk of the controller code is in the login method, which gets a new authentication token based on the user input and attempts to authenticate against the Spring AuthenticationManager. If this attempt fails, an AuthenticationException will be thrown, letting us know that the user login or password does not match the credentials in our user store.

The final bit of code is the security definition itself, which is shown in **Listing 8**. You can use any authentication system you want including LDAP, OAuth, SAML, or even a custom authentication provider. For the purpose of this example, we are going to use a simple authentication provider with plain-text passwords

LISTING 7 LISTING 8

```
public class LoginController implements DialogController {
  @Autowired
  private AuthenticationManager authenticationManager;
  private ScreensConfiguration screens;
  private FXMLDialog dialog;
  public void setDialog(FXMLDialog dialog) {
    this.dialog = dialog;
  }
  public LoginController(ScreensConfiguration screens) {
    this.screens = screens;
  }
  @FXML Label header;
  @FXML TextField username;
  @FXML TextField password;
  @FXML  void login() {
    Authentication authToken =
new UsernamePasswordAuthenticationToken(
username.getText(), password.getText());
    try {
      authToken =
authenticationManager.authenticate(authToken);
      SecurityContextHolder.getContext().setAuthentication(
authToken);
    } catch (AuthenticationException e) {
      header.setText("Login failure, please try again:");
      header.setTextFill(Color.DARKRED);
      return;
    }
    dialog.close();
    screens.showScreen(screens.customerDataScreen());
  }
}
```

Download all listings in this issue as text

ANNOTATION/INTERFACE	SPRING EQUIVALENT	RECOMMENDED USAGE
@Inject	@Autowired	@Inject LETS YOU DO DI ON VARIABLES OR CONSTRUCTOR ARGUMENTS AND IS A DROP-IN REPLACEMENT FOR @Autowired.
@Named	@Component	ELEMENTS THAT CAN BE INJECTED ARE MARKED AND NAMED BY USING THE @Named ANNOTATION ON A CLASS.
Provider	FactoryBean	THE Provider INTERFACE MARKS A FACTORY CLASS THAT CAN CREATE AN IMPLEMENTATION WHEN THE get METHOD IS CALLED.

Table 1

LISTING 9 LISTING 10

```
@Secured("ROLE_MANAGER")
public void remove(Customer customer) {
  restTemplate.delete(
"http://localhost:8080/crm/customer/" + customer.getId());
  customers.remove(customer);
}
```

Download all listings in this issue as text

embedded in our Spring configuration file.

Warning: You probably know this, but never, ever use a plain-text security provider in a production application, because all your passwords will be exposed to anyone who opens your JAR file on the client. Now, choosing good passwords is another matter, but at least our employee has a sense of humor about his password being exposed.

Once this is set up, locking down methods in your application so the information can be accessed only by a particular user role is as easy as adding a Spring Secured annotation, as shown in **Listing 9**.

To demonstrate how security works in practice, **Listing 10** shows the UI definition of the Remove button, which will delete the selected customer from the table when the action handler is invoked.

Notice that an AccessDenied Exception is thrown when the wrong user is logged in. We catch it here and display the error dialog box we created in Part 1 of this series.

Dependency Injection Based on JSR 330

Up to this point, we have been using Spring's configuration and DI mechanisms, but what if we want the implementation to be more portable? JSR 330 provides a standard DI system that works across different providers, including Spring, Guice, or any Java EE 6 server. However, to take advantage of JSR 330, we need to do some refactoring of our application code.

First, it is helpful to understand the core JSR 330 annotations and how they relate to the Spring annotations we have been using. The JSR 330 annotations we will need to use are shown in **Table 1**.

We are going to use these annotations to replace the Spring annotations to replace the Spring JavaConfig that we used in Part 1 with a standards-compliant implementation.

Note: The full source code for the converted application can be found here.

To use JSR 330 annotations with your Spring application, two steps are required to set up your application:

1. Import the javax.inject and spring-context libraries. You can find the full Maven dependencies for this in the project pom.xml file.
2. Add the @ComponentScan annotation to your configuration file. You also need to specify a basePackage with a full annotation definition such as the following:

```
@ComponentScan(
basePackages=
"steveonjava.client")
```

Now it is simply a matter of converting the Spring annotations to their equivalent JSR 330 annotations, with some refactoring to take advantage of the different model.

Here is how I converted the different UI and model abstractions:

Controllers. The controller classes were already well abstracted. Rather than having them created using Java Config, I added the @Named annotation to each so they could be instantiated by the container based on type. Also, I converted all the @Autowired annotations to @Inject.

As a little bit of cleanup, I removed the base class to set the Dialog/Stage and instead used a little JavaFX trick to get access to the Stage from the root FXML element:

```
((Stage) root.getScene().
getWindow()).close();
```

Dialog boxes. The Spring JavaConfig provided a nice mechanism for creating multiple instances of the FXMLDialog from a single class, but since JSR 330 uses class-based annotations, I broke these definitions out into separate subclasses that extend FXMLDialog and have the @Named annotation for lookup by type.

Another issue was getting lazy creation and separate UI instances for the dialog boxes upon invocation. Without this, the dialog boxes retain information between usages (for example, when you reopen the Add Customer dialog box, it erroneously remembers the last user entered). This is a perfect use case for JSR 330 Providers, so I refactored the FXMLDialog and subclasses to be Providers of JavaFX Stage objects.

Listing 11 shows the full source code for the new FXMLDialogProvider class.

Notice that the FXML loading code has also been updated to create controllers based on the type specified in the FXML file. This avoids the dependency on Spring JavaConfig to create the controller classes, allowing us to delete the ScreensConfiguration file altogether.

With these changes, we are able to get rid of all the Spring DI dependencies except for the following:

- Context loading. You still need to bootstrap Spring in your start method using the AnnotationConfigApplicationContext.
- Controller instantiation. JSR 330 makes no provision for dynamic loading of objects by type, so this requires a provider-specific extension such as applicationContext.getBean (Spring) or injector.getInstance (Guice).

NEW CONTROL

The TableView is one of the most important **new controls introduced in JavaFX 2**, and it is central to any data-based application.

LISTING 11

```
public class FXMLDialogProvider implements Provider<Stage> {
  @Inject
  private ApplicationContext applicationContext;
  private final StageStyle style;
  private final URL fxml;

  public FXMLDialogProvider(URL fxml, StageStyle style) {
    this.style = style;
    this.fxml = fxml;
  }

  @Override
  public Stage get() {
    Stage dialog = new Stage(style);
    dialog.initModality(Modality.WINDOW_MODAL);
    dialog.initOwner(CustomerApp.getPrimaryStage());
    try {
      FXMLLoader loader = new FXMLLoader(fxml);
      loader.setControllerFactory(
new Callback<Class<?>, Object>() {
        @Override
        public Object call(Class<?> aClass) {
          return applicationContext.getBean(aClass);
        }
      });
      dialog.setScene(new Scene((Parent) loader.load()));
    } catch (IOException e) {
      throw new RuntimeException(e);
    }
    return dialog;
  }
}
```

Download all listings in this issue as text

We are still using RestTemplate and Spring Security, but this doesn't limit our portability to a different DI container for our application.

Gloves-Off Conclusion

I have shown what you can do by leveraging some of the best features of Spring to improve your application, and have even taken it a step further by demonstrating how you can simultaneously make your application standards-compliant. However, every technology you add to your application increases the complexity, so is it really worth the investment?

If you are already using Spring on the server side, this is a slam dunk. You can leverage your existing Spring knowledge on the client, reuse model objects on both sides of your codebase, and be well equipped to handle the occasional Spring bug or tricky configuration issue. It is also likely that with your Spring expertise, you will find a multitude of ways to optimize and improve your application beyond what I have discussed here.

On the other hand, if you have a big investment in a Java EE back end, you might want to keep your dependency on Spring in the client to a minimum. In this case, sticking to JSR 330 standard annotations will give you more freedom to switch to another DI container, such as Guice, in the future. Also, you might want to use a different REST library, such as Jersey, so you can reuse code you use in your server-side unit tests to fetch data for your client.

If you are not heavily invested in a DI container on the server side, you should consider whether the benefits outweigh the costs. The following are a few of the hurdles you will need to cross to implement Spring on the client:

- Many JAR files and dependencies. My final project ended up with 21 JAR files and 5 MB of extra goodness. This increases the download size of your application, which matters more on the client where users have to wait for the full application to download before opening it.
- Cryptic error messages. Spring (and any DI system) adds an extra level of complexity that often masks error messages coming from your code. Expect at least two or three levels of indirection before you find the root cause and an occasional gap where the true error condition is not reported.
- Startup time and initialization order. Server-side systems are typically optimized for high throughput with little regard to startup time or order of initialization. DI takes away a lot of the control on initialization order and makes startup costs harder to track.

These issues might not matter for your use case or application profile, and they will likely become trivial as the complexity of your application goes up. Conversely, many of the benefits of DI are truly realized only as the complexity and team size of your project go up, so if you expect your application to grow and have a long maintenance cycle, the up-front investment in leveraging a framework such as Spring on the client can pay off hugely in the long run. `</article>`

LEARN MORE

- "JavaFX in Spring, Part 1"
- Stephen Chin's blog

MAKE THE FUTURE JAVA

Java™

FIND YOUR JUG HERE

One of the most elevating things in the world is to build up a community where you can hang out with your geek friends, educate each other, create values, and give experience to you members.

Csaba Toth
Nashville, TN Java Users' Group (NJUG)

LEARN MORE

ORACLE®

JOSH JUNEAU

BIO

Part 1

Jython 101—A Refreshing Look at a Mature Alternative

Learn Jython and take advantage of the Python language syntax.

Alternative languages for the Java Virtual Machine (JVM) offer different options for developers, enhancing productivity and enabling the use of different coding techniques. One of the first alternative languages available for the JVM was Jython. As the name implies, it allows developers to utilize the Python language syntax and apply coding patterns that are used with CPython, the canonical implementation of Python written in the C language.

Learning Jython can be advantageous because different ports of Python can be found across various platforms, not solely on the JVM. The mantra "learn once, apply everywhere" applies to Python, which makes it a handy tool for any developer's arsenal.

Jython not only allows developers to utilize Python on the JVM, but it also allows the use of many Python APIs and frameworks. Thus, popular technologies from the Python realm, such as the Django Web framework, can be utilized on the JVM. Applying the Python syntax to Java APIs can also help make developers more productive. For instance, writing a Swing or JavaFX application using Jython can eliminate dozens of code lines.

This article, the first in a two-part series, provides a brief overview of the language, demonstrating syntax and techniques that offer productive alternatives to the Java language.

Variables and Expressions

We'll begin with the most basic principle of many programs: declaring variables that can be used to work with data. Jython variables are not restricted to a single type; therefore, declaration of a variable is as simple as providing an identifier name and a value to assign.

The name is used to identify the object that is assigned to the variable, not to create a variable with a type designation. As such, this allows variables to remain untyped, meaning that a variable's type can change at any time after it has been declared.

Let's take a look at two simple examples. First, a variable identified as x is assigned the value of 0. Second, a variable identified as y is assigned a string value.

```
>>> x = 0
>>> y = 'Java Magazine'
```

Because Jython variables are not bound to a certain type, their types can change at any time, as shown below:

```
>>> x = y
>>> print x
Java Magazine
```

This example demonstrates some basic features of Jython: using expressions and how to print output to the terminal. Jython allows developers to use expressions easily. In fact, Jython can be a great calculator by simply firing up the interactive terminal, entering the expression that you wish to evaluate, and pressing the Return key.

The expression is evaluated immediately, or to be more formal, a print statement can be utilized to display the output. The print statement is analogous to the System.out.println() statement in Java, and anything enclosed in single or double quotes is treated as a string value. Expressions used with the print statement are first evaluated and then printed:

```
>>> print 3 + 4
7
```

Note that variables with identifiers that begin with a single underscore are treated as private variables in a class context.

Conditionals and Iteration

Conditionals in Jython can be written more concisely than their Java counterparts. An if statement, for example, does not require parentheses or braces to enclose the different blocks. Rather, a Jython if statement has the following syntax (interactive interpreter characters have been removed within the statement so you can see the alignment):

GOOD OPTIONS

Alternative languages for the JVM offer different options for developers, enhancing productivity and enabling the use of different coding techniques.

```
jython = 'fun'
if jython == 'not fun':
    print 'Jython is not fun'
elif jython == 'sort of fun':
    print 'Jython = sort of fun'
else:
    print 'Jython is fun!'
>>> Jython is fun!
}
```

This example demonstrates many syntactic differences between Jython and Java. The first rule of thumb is that alignment is the key. That is, rather than using brackets to separate blocks of code, each block must be aligned uniformly, or a compiler error will be raised. This is a benefit of using Jython because it is much easier to maintain code that is organized and easy to read, and there are very few unnecessary punctuation marks sprinkled throughout code.

The developer can decide upon the number of spaces that are used to differentiate blocks of code in Jython, but the same number of spaces must be used to indent every line of code within that block. Therefore, in the if statement example above, we see that four spaces (the Python language standard) are used to differentiate each block that comprises the statement.

Keeping this in mind, we can apply the same coding structure to iteration. Jython contains iteration techniques that are similar to those found in Java. For instance, you can create an indefinite loop using a while loop or a bounded loop using a for loop. Let's take a look at a couple of examples.

First, let's review how the while loop is coded in Jython. This type of loop will continue to process while the specified condition evaluates to True, just as in the Java language. The example below demonstrates a simple while loop that will be iterated over until the counter variable value reaches a specified limit. Note that the body of the loop is indented by four spaces.

```
>>> counter = 0
while counter <=3:
    print counter
    counter += 1
...
0
1
2
3
```

The Jython for loop uses a similar coding structure. The following example demonstrates the same example, but this time with a for loop.

```
for counter in range(4):
    print counter
...
0
1
2
3
```

The Jython for loop is very similar in context to the Java for loop that was introduced in Java 5.0 in that it is used for iteration over collections of data, rather than being a counter-based iterator. In the example above, a variable named counter is printed out four times, once for each value within the range of 0 to 3.

Ranges and Data Structures

The preceding example demonstrates the range() function, which returns a list of numbers that reside within a given range. The range() function accepts one to three arguments, the first being the range starting point, the second (optional) being the end of the range, and the third (optional) being the numeric increment for each step in the range. If the second argument is left off, as in the preceding example, the range will use 0 as the starting point and the given argument as the ending point.

Below is another simple range example that demonstrates the use of each argument. In this example, each number in the range between 5 and 25 is printed, stepping in increments of 5:

```
>>> range(5,25,5)
[5, 10, 15, 20]
```

Given that the range() function returns a list of values, the for loop is a perfect candidate for iterating over the returned result. Jython contains a number of different data structures that can be used to work with collections of data: lists, dictionaries, sets, and tuples.

Perhaps the most commonly used container is the *list*, a sequence type whose objects can comprise any Jython type. Moreover, various objects of different types can be contained within the same list. Another powerful capability of a list is that it can be dynamic, meaning that it can change over time, as needed. No static length needs to be specified at the time of declaration, and declaring a list is as simple as assigning an empty set of brackets to an identifier, as follows:

```
>>> my_list = []
```

Most lists do contain values, of course, and to create a populated list, assign a sequence of different Jython objects to an identifier, as follows:

```
my_list2 = [1,3,'five',7,'nine']
```

Note that the list in the previous example contains a mixture of integers and strings. To add values to an existing list, either call the append() method to insert the new value at the end or call the insert() method to insert the value in a specified location, as follows:

```
>>> my_list.append(2)
>>> my_list.insert(1,'four')
>>> my_list
[2, 'four']
```

The insert() method accepts two arguments, the first being the position at which to insert the value and the second being the value to insert. List indices begin with 0, so to insert a value into the second position you would use the position of 1, as shown in **Listing 1**.

Lists can be handy for transporting data within an application, and there are a handful of different methods available (extend, pop, remove, del) for working with the data. The code in **Listing 2** shows the use of these methods.

Of course, lists are accessible via indices, which provide the ability to retrieve values in a variety of ways.

Dictionaries are mapping types that allow a developer to provide a key/value pair of any datatype for each item.

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

```
>>> lang_list = ['Java','Jython']
>>> lang_list.insert(1,'JRuby')
>>> lang_list
['Java', 'JRuby', 'Jython']
```

Download all listings in this issue as text

Listing 4 demonstrates the use of a dictionary containing book titles and their associated star ratings.

Sets are unordered collections that contain no duplicates. They cannot contain mutable data, but the sets themselves can be mutable. In order to use a set, you must import it from the sets module. **Listing 5** demonstrates some examples of working with a set.

The final data collection type is a *tuple*. A tuple is very similar to a list, except that it is not mutable. Tuples are usually populated with heterogeneous values, such as a parameter list. Tuples are demonstrated in **Listing 6**.

List comprehensions are a powerful way to apply an expression or function to every element within a list. List comprehensions are a great example of a way in which Jython can make a Java developer's life easy. To write a list comprehension, simply iterate over a list and apply the expression or function to each element using the syntax [<expression or function> for <identifier> in <list>]. **Listing 7** contains some simple examples.

Functions and Lambdas

Functions are one of the most powerful features of Jython. Any functionality that can be used more than once should be placed into a function. Functions in Jython are similar to Java methods that return a value, with a few syntactic differences.

Jython function signatures are very cursory compared to Java method signatures. The def keyword is used to define a function, and unlike Java, functions are executable statements in Jython. As such, this means you can nest functions, pass them as parameters to other functions, and so on. There is no requirement to specify a return type because Jython functions can return one or more values. Further, there is no type declaration required for any parameters.

Listing 8 contains an example of a function that returns a copy of a given string with every other letter capitalized. If a number is passed as the argument to the function, a number with the same value is returned.

Default values can be defined for functions in Jython. For example, the following function multiplies two numbers and returns the result. However, the second parameter contains a default value of 1. If a single number is passed to the function in the example, it will be multiplied by the default value and the same number value will be returned.

It is possible to write a function in Jython that takes only one line of code. Such a function is known as a *lambda*. Lambdas are anonymous functions that can be evaluated and return a result within one line. The following is an example of a lambda that returns the square root of a given number.

```
squared = lambda x: x * x
```

In the example above, the x represents the value passed into the function and also the returning value. When a value is passed to the lambda, the expression on the right side of the colon is evaluated, assigning the passed-in value to the x variable; then the expression result is assigned to the variable on the left side of the colon and returned. For instance, this lambda can be called as the following lines demonstrate:

```
>>> squared(3)
9
```

Functions in Jython can accept any number of parameters by prefacing the parameter with a * character. An argument that is preceded with * indicates that the parameter value will contain a sequence of zero or more. This means that a function can be written as in the following example, accepting an arbitrary number of parameters:

```
>>> def my_func(*args):
...     return len(args)
...
# Calling my_func with
# an arbitrary parameter set
>>> my_func()
0
>>> my_func(1,2,3)
3
```

Functions can also accept key/value lists of data, such as dictionaries, so long as the parameter is

LISTING 6 / LISTING 7 / LISTING 8

```
>>> lang_tuple = ('Java','Jython')
>>> lang_tuple
('Java', 'Jython')
```

Download all listings in this issue as text

preceded with **. The following demonstrates this behavior:

```
>>> def f(**kwargs):
...     for key in kwargs:
...         print key
...
>>> f(p1="1", p2="2", p3="3")
p2
p1
p3
```

Decorators are a syntactic sugar in Jython that allows one function to transform another, thereby enhancing the actions of a function that is being decorated. To define a decorator, create a function that accepts another function as a parameter, and then call the function that was passed in as a parameter and use it to perform some task. You can then use a special syntax (@function_name) to apply that decorator function to another function, which will, in turn, cause the decorated function to be passed into the decorator.

Listing 9 demonstrates an example of a decorator function and a function that is decorated. The decorator function contains an inner function that squares the sum of the values accepted in the decorated function.

WRITE LESS CODE

Applying the Python syntax to Java APIs can also help make developers more productive. For instance, writing a Swing or JavaFX application using Jython can eliminate dozens of code lines.

Classes

Jython class syntax is very similar to that of Java, although there are a handful of differences in concept. For instance, Jython classes support multiple inheritance—they can extend more than one base class. Classes, just like other objects in Jython, are dynamic and can be changed at runtime.

Classes contain instance attributes, such as data attributes and methods. Data attributes are analogous to instance variables in a Java class, and class methods are functions that are bound to an object. **Listing 10** contains a class representing a Car object. The Car class contains three data attributes and one method.

Some method identifiers begin with two underscores and end with at least one underscore. Such methods are mangled at runtime, so direct access is not possible. These methods are also known as *magic methods*. __init__(), one such method, is analogous to the class constructor in Java. It is invoked upon instantiation or execution of a Jython class.

The following lines show how to create a new Car object and then invoke the print_car() method:

```
>>> c = Car('ACar','Jy','2005')
>>> c.print_car()
ACar Jy – 2005
```

In the preceding example, the parameters are passed into the __init__() method of the Car class upon instantiation.

Platform Advantage

Another important feature of Jython is its ability to utilize all libraries that the Java platform has to offer. Over the years, the Java platform has grown in power as more libraries and APIs have been added to increase functionality and enhance developer productivity. Along the way, many developers have created external libraries

LISTING 9 LISTING 10

```
# Decorator function that returns the square
# of the sum of the parameters that are passed
# into the decorated function
def squared(func):
  def inner(*args):
    val = func(*args) * func(*args)
    return val
  return inner

# Decorated function that accepts an arbitrary number
# of parameters and sums them together.
@squared
def square_total(*nums):
  idx = 0
  total = 0
  while idx < len(nums):
    total = total + nums[idx]
    idx = idx + 1
  return total
```

Download all listings in this issue as text

that can be imported into a Java project to add functionality.

If you are using Jython, you can use all the libraries that come standard with the Java platform. Moreover, many third-party libraries can be used within Jython projects by simply adding their associated JAR files to the CLASSPATH and importing the required classes within the code. Let's take a look at how to use some Java libraries that are part of the platform.

In **Listing 11**, java.util.ArrayList is imported into an interactive session. Note how the class is imported and how the new keyword is not required when instantiating a new instance of ArrayList. The Jython syntax makes working with Java objects very easy.

MANY PORTS
Learning Jython can be advantageous because different ports of Python can be found across various platforms.

This is just the tip of the iceberg. You can create just about any Java application using the Jython language. As a matter of fact, **Listing 12** demonstrates how to create a simple "Hello World" application using JavaFX 2.2 and Jython. The example introduces quite a few concepts with regard to Java and Jython integration, for example, it is possible to extend a Java class with a Jython class, and calls to Java getters and setters are implicit. Java and Jython integration is a huge topic, and Part 2 of this series will further address the topic.

Commenting Your Work

So how do you place comments within your work? A pound (#) symbol indicates the start of a comment. If it is used on the first column position in a line, the entire line can be a comment, as follows:

```
>>> # This is my variable
>>> my_var = 'me'
```

Comments can begin at any position within a line, so you can also add a comment after a line of code, as follows:

```
>>> another_var = 7 # variable
```

Docstrings begin and end with a series of triple single-quote characters. A docstring can be used to document code and register it with the Jython help system. A docstring is started directly after a function signature. You can also call a function's __doc__ method to return any docstring that has been associated with it. An example of using docstrings can be seen in **Listing 13**.

LISTING 11 LISTING 12

```
>>> from java.util import ArrayList
>>> arr = ArrayList()
>>> arr.add(1)
True
>>> arr.add(2)
True
>>> arr
[1, 2]
```

Download all listings in this issue as text

Conclusion

The JVM offers a mature platform for application development. The Java language itself is full-featured and supplies developers with the ability to create sophisticated solutions. However, there is always room for alternative languages that will add benefit to a developer's toolkit.

Java is a type-safe language that offers substantial space for dynamic languages on the JVM. Alternatives such as Jython, JRuby, and Groovy offer dynamic capabilities and advanced features that can help developers create solu-

LISTING 13

```
>>> def hello_world():
...   ''' This function prints hello world
...     to the terminal '''
...   print 'Hello World'
...
>>> hello_world()
Hello World
>>> help(hello_world)
Help on function hello_world in module __main__:

hello_world()
    This function prints hello world
    to the terminal
>>> hello_world.__doc__
' This function prints hello world\n     to the terminal '
```

Download all listings in this issue as text

tions in an entirely different way. Such alternatives have allowed the Java ecosystem to grow into an even more desirable platform for development.

Jython brings advanced features such as dynamic variables and lambdas to the JVM, along with a full-featured set of libraries and APIs. Moreover, it is possible to develop an application using Jython and change little or no code to run it on other platforms using CPython (C Language) and IronPython (.NET).

Here, we barely scratched the surface of what Jython has to offer. In Part 2, we will cover Java and Jython integration in detail and move on to some advanced topics. `</article>`

LEARN MORE

- Jython Website
- *The Definitive Guide to Jython* (Apress, 2010)
- Jython Wiki
- Jython Monthly newsletter

MAKE THE FUTURE JAVA

Java™

FIND YOUR JUG HERE

CEJUG is celebrating our 10-year anniversary in 2012! We follow Java technology with passion, share knowledge with pleasure, and create opportunities for students and professionals with ambition.

Hildeberto Mendonça
The Ceará Java User Group (CEJUG)

LEARN MORE

ORACLE®

VIKRAM GOYAL

BIO

A Common Advertising Platform for Java ME Developers

Learn how to create a platform for inserting advertising within your applications.

You can pay for your development efforts for a killer app by doing either of the following:

- Charging an arm and a leg for the app
- Displaying advertising within the app

In this day and age, where developers frequently give away apps using the freemium model, you would be hard-pressed to use the first option. Thus, putting advertising within your application is a better way to get credit for your work. Then, if your application is really successful, you might start charging for an ad-free option.

In this article, I'll guide you through an *ad-injection* process whereby you can display ads in defined spaces. The example code will help you create and define your own customized ad-injection processes. I won't cover actual advertising APIs, which do not differ in the way they supply ads. The documentation about how to use such APIs is simple and clear and can be seen on the respective Websites.

Ad Injection

Ad injection is a simple process, similar in functionality to Web advertising, but the implementation requires careful planning due to the lack of Web-based modules.

Unlike with Web-based advertising, try to think of ad injection as altering the fundamental display of the running MIDlet to modify, display, and create interactive ads that do not affect the way the user interacts with the main application. The ads are "injected" into the display at the top, bottom, right, or left side of the main application and, indeed, over the entire display for the main application. In the following sections, I will discuss the mechanics of an ad-injection platform and develop one as well.

The Ad-Injection Process

Since there are only two types of displayable units that a MIDlet can use at the top level (the Form and the Canvas), our job is easy, because we need to learn to manipulate only these units.

The ad-injection process must be kept separate from the main MIDlet classes and code. The MIDlet classes and code should be semi-aware of the presence of advertising, but other than that, the whole ad-injection process should be controlled by the ad-injection package. The MIDlet only needs to identify the basic parameters of advertising—for example, the participation in the advertising program, the frequency with which ads should be displayed, where the ads could be displayed, and so on. Other than that, the MIDlet should be completely independent of the ad-injection process and it should let the specialist classes of the ad-injection process perform the negotiation between the client and the ad server, modify the display, and determine availability.

The org.adinjection Package

I have created four classes/interfaces in this package that encapsulate the ad-injection process: AdItem, Advertisable, Injector, and

PHOTOGRAPH BY JONATHAN WOOD/GETTY IMAGES

Supplier. I will discuss these in detail next, one by one.

AdItem. AdItem represents a unit of advertising. It's an independent unit that extends the ImageItem class and implements the ItemCommandListener interface, as shown in **Listing 1**, so that clicks on this unit can be recorded and handed over to the right API. Think of it as a banner in traditional Web-based advertising.

As you can see from the code in **Listing 1**, the AdItem class encapsulates everything that can be expected from traditional banner advertising.

Advertisable. The Advertisable interface defines the properties that a MIDlet must possess to be able to participate in the ad-injection process (see **Listing 2**). By implementing this interface, the MIDlet says to the ad-injection process that it is ready to participate in the process based on the guidelines that it defines.

There are some very basic concepts that this interface defines. Therefore, each MIDlet must implement the interface and define the following methods.

- initAdvertising(): This method is where each MIDlet can decide whether to participate in the ad-injection process and create an Injector (defined later). This is also where a MIDlet can define the frequency with which ads are changed.
- destroyAdvertising(): When you have had enough advertising, or the user decides to upgrade to an ad-free full version of your application, you can use this method to destroy all ad-injection processes.
- getTargetDisplay(): This method defines the target display that can be manipulated to inject the ads.
- receivesAds(): This method allows you to temporarily stop the display of advertising. Return false if you want to stop ads based on application logic.
- getCompatibleAdTypes(): This method controls what types of ads—overlay ads, text ads, image ads, or all sorts of ads—are displayed.
- getCompatibleAdLocation(): This method specifies compatible locations for ads, for example, top, bottom, or "it doesn't matter."
- getPreferredAdWidth() and getPreferredAdHeight(): You might not want the ads to be displayed across the entire width or height of your application. Using these methods, you can restrict the width and height that are used for ads.

Injector. Injector is the main class of the org.adinjection package (see **Listing 3**). This class takes a target MIDlet and manipulates its display with ads that are sourced from the Supplier class (discussed next).

This class is the go-to layer between your application MIDlet and the advertising APIs. It is separate from your MIDlet, and it doesn't deal with the headache of getting the ad or tracking the clicks on ads. It just assumes that the ad will be supplied to it (based on

LISTING 1 LISTING 2 LISTING 3

```
public class AdItem extends ImageItem implements
ItemCommandListener {
  private String click_url;
  private boolean overlayAd;
  public AdItem(
    String label, Image img, int layout, String altText,
String click_url) {
    super(label, img, layout, altText);
    this.click_url = click_url;
    this.overlayAd = false;
    Command openCommand = new Command("Open", Command.ITEM, 1);
    addCommand(openCommand);
    this.setItemCommandListener(this);
  }

  public void setOverlayAd(
boolean overlayAd) { this.overlayAd = overlayAd; }
  public boolean isOverlayAd() { return this.overlayAd; }

  public void commandAction(Command c, Item item) {
    // This is where the click-through action will take
    // place. That is, the user has shown interest in the ad
    // by clicking on it. For the overlay ad, the user could
    // have been just clicking to close it.
    System.err.println("Clicked"); // call the click_url
  }
}
```

Download all listings in this issue as text

what it knows about the MIDlet) and then displays the ad in the MIDlet by manipulating the screen.

When the Injector class is created, it creates and initializes a separate running thread. It also sets the time delay between changing the ad.

The initAndDisplay() method shown in **Listing 4** simply puts the current thread to sleep for the predetermined interval and then calls the inject() method. The flag is set by the calling MIDlet if it wishes to stop participating in the ad-injection process.

Finally, the inject() method does the magic of figuring out whether the target displayable item is a Form or Canvas, whether the ad needs to be displayed at the top or bottom (or as an overlay ad), and what the perfect ad dimensions should be, as shown in **Listing 5**. I have left manipulations on the Canvas side for you to try out.

The code looks more complex than it really is. The different parameters have to be taken into account, and the display needs to be modified. In other words, the Form needs to inject an ad either at the top or the bottom of the display, and it needs to determine whether the ad is being displayed for the first time or being repeated.

Supplier. The final class, Supplier, is an encapsulation of the advertising API. It gets the ad based on the physical location, demographic details, and a host of other factors. It also resamples the ad to make it fit within the preferred width and height of the target display.

The code in **Listing 6** pretends that it is connecting to a remote server and shows two different ads randomly each time it is called. In the code, I have loaded an image directly from the file server, although in real life, this would be done by an API call over a network. Over 2G networks, loading an image might take a while.

To avoid bottlenecks resulting from slow network speeds, I highly recommend that you encapsulate all network calls in a separate thread. Although the whole ad-injection process happens within its own thread, networking should always have its own thread to compensate for slow and unreliable networks.

BEFORE YOU TEST

When working with live ad platforms, it is advisable that you get your MIDlet signed before testing (and, of course, before actually making your MIDlet live).

LISTING 4 / LISTING 5a / LISTING 5b / LISTING 6

```
private void initAndDisplay() {
 while(flag) {
  try {
   Thread.sleep(this.adInterval);
  }
  catch (Exception ex) {
   ex.printStackTrace();
   return;
   //
  }
  inject();
 }
}
```

Download all listings in this issue as text

An Example MIDlet

To put this all together, we need an example MIDlet (see **Listing 7**). The AdvertisingOptionsMIDlet is just that. It is not a special MIDlet, but it implements the Advertisable interface and, therefore, it has to define the methods and properties of that interface so they can be applied to the MIDlet itself. The MIDlet is Form-based in accordance with our Injector and can handle only Form-based displays at the moment.

As seen in **Listing 7**, we specify that the ads are never more than one quarter of the height of the actual form, that the ad width can be the width of the form, that there is at least a five-second delay between different ads, and so on.

SOME ADVICE

To avoid bottlenecks resulting from slow network speeds, I highly recommend that you **encapsulate all network calls in a separate thread.**

Conclusion

This article described a simple way for Java ME developers to create a platform for inserting advertising within their applications. In this simple approach, the MIDlet is not cluttered with the advertising code; instead, it contains only the application logic. The advertising platform is solely responsible for negotiating with an advertising API and creating the advertising mechanism. The MIDlet stays independent.

You can use the code presented in this article to develop something robust that actually negotiates with a real, live advertising API. Most advertising APIs work the same way, and by creating your own unique platform, you can be independent of the different networks and yet plug in the network of your choice.

When working with live ad platforms, it is advisable that you get your MIDlet signed before testing (and, of course, before actually making your MIDlet live). Getting your MIDlet signed will help prevent nasty security messages from continuously annoying your end users. Signing will also make your application more trustworthy to end users. </article>

LEARN MORE

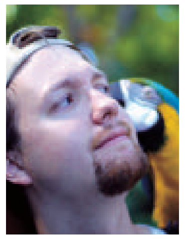
- MIDlet class
- Java ME

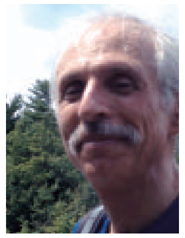
LISTING 7

```
// method implementations mandated by Advertisable interface
// the display that needs to be modified
public Display getTargetDisplay() {
  return this.display;
}

// some logic here could determine if ads should be stopped
public boolean receivesAds() {
  return true;
}

public int getCompatibleAdType() {
  return IMG_AD_TYPE;
}

public int getCompatibleAdLocation() {
  return BOTTOM_AD_LOCATION;
}

public void initAdvertising() {
  injector = new Injector(this, 5000);
}

public void destroyAdvertising() {
  injector.destroy();
  injector = null;
}

public int getPreferredAdWidth() {
  return mainForm.getWidth();
}

public int getPreferredAdHeight() {
  return (int)mainForm.getHeight() / 4;
}
```

Download all listings in this issue as text

//fix this /





In the January/February 2013 issue, Simon Ritter gave us a code challenge around the Connected Device Configuration (CDC) and the Connected Limited Device Configuration (CLDC), which are used in Oracle Java ME Embedded Client and Oracle Java ME Embedded. He gave us a code sample and asked how it can be modified so that all messages are read in a single get(Data) call.

The correct answer is #4. It will reset the length of the datagram to 256 bytes, thereby guaranteeing that it is big enough to hold the maximum-length message. Because we reuse the datagram, it is also the most efficient solution and best suited for a resource-constrained, embedded device.

Answer #1 is wrong because this has no bearing on the way the length of the datagram is handled. Answer #2 would solve the problem because it would create a new instance of the datagram for every receive. However, in embedded systems, the emphasis should be on memory and processing efficiency so this answer is, strictly speaking, incorrect in this context. Answer #3 will reset the read pointer of the datagram back to the start but will not affect the length of the datagram.

This issue's code challenge comes from Jason Hunter (top left), author of *Java Servlet Programming*, 2nd Edition (O'Reilly Media, 2001) and deputy CTO at MarkLogic, and Boris Shukhat, applications programming manager, vice president, Bank of America Merrill Lynch.

1 THE PROBLEM

The Java Persistence API (JPA) has some undocumented surprises in how it handles parameterized SQL and the result set.

2 THE CODE

A junior programmer was asked to add a parameterized SQL query and its execution to a program using JPA. The code looked like this:

```
EntityManager em = ...;
String sql = "select department from emp where fname=? and mname=?
and lname=?";
Query query = em.createNativeQuery(sql);
String[] name = new String[3];
for (int i=0; i<3; i++) query.setParameter(i, name[i]);
List departments = query.getResultList();
if (departments == null)
      System.out.println("Not available");
else
      for(String department : departments)
            System.out.println(department);
```

The code did not work.

3 WHAT'S THE FIX?

1) Use createQuery() instead of createNativeQuery().

2) Put parameter values in single quotes.

3) Change the order of how parameters are set.

4) Check the size of the departments list.

GOT THE ANSWER?

Look for the answer in the next issue. Or submit your own code challenge!

ART BY I-HUA CHEN