Java magazine

By and for the Java community

//JULY/AUGUST 2013 /

JAVA

@java

With Java at its core, Twitter supports more than 400 million Tweets per day #performance

10:51 AM - 1 August 13

+ 11 JAVAONE PREVIEW | 23 ORACLE TEAM USA | 28 KIRK PEPPERDINE ON PERFORMANCE TUNING

ORACLE.COM/JAVAMAGAZINE

//table of contents /

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
Intern
Curran Mahowald

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

Get Java EE Expertise

—Oracle University—

✓ New Java EE 7 Training

✓ Engineering-Developed Courses

✓ Aligned with Java EE Certifications

✓ Experienced Java Instructors

Register Now

ORACLE®

Many of us in the Java community are competitors. I'm a runner who participates in an occasional half marathon. While my overall goal is always to finish each race, I also want to do better—to beat my last time and get a new personal best. It's human nature. We all want to keep getting better.

In this issue, we dive into the topic of performance. In our cover story, we look at how migrating its core infrastructure to the Java Virtual Machine (JVM) enabled Twitter to beach the fail whale and support more than 400 million Tweets per day.

Performance is also a huge issue for ORACLE TEAM USA, which is the defender in the 34th America's Cup, taking place in San Francisco Bay in September 2013. The team relies on a custom-built Java data collection and information delivery system that is used for designing and developing the boat as well as for providing information to get better performance on the racecourse.

If you're developing Java applications, making them run better every day is definitely a priority. We check in on the topic of Java performance tuning with Kirk Pepperdine, a leading expert in this area, who weighs in on the effect of cloud computing, changes in the JVM, multicore processors, and more.

Finally, JavaOne San Francisco is nearly here. Have you made your travel plans? If you need motivation, check out our JavaOne preview, where we interview four JavaOne speakers about what they're looking forward to at JavaOne. Plus, we give you information to help you make your JavaOne experience educational, inspirational, and fun. I hope to see you there!

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®

FOCUS ON JAVA EE 7, EMBEDDED

More than 2,000 developers and IT professionals attended JavaOne Russia, the biggest Java conference in Russia, which took place April 23–24 in the majestic Crocus Expo Center.

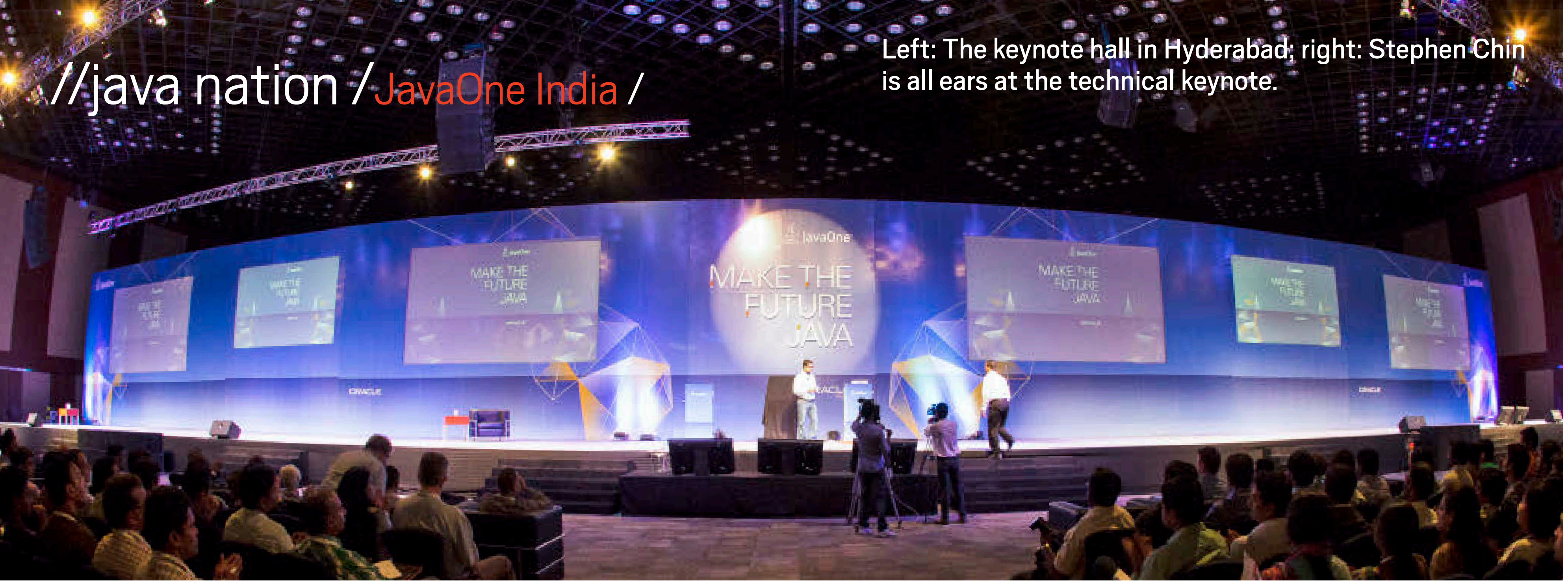
With its upcoming release, Java EE 7 got a lot of buzz. There were several sessions and labs about the platform, JSON Processing, HTML5, batch applications, Hadoop, modular applications in the cloud, WebSocket, Java Message Service, and more. Oracle's **Anil Gaur** told attendees that "187 experts, 32 companies, and 16 spec leads from the Java ecosystem" contributed to the Java EE 7 release, a developer-friendly release that can be used to develop everything from lightweight apps to enterprise apps.

Java embedded was another hot topic. "Today we are at a tipping point," explained Oracle's **Nandini Ramani**. "The number of devices far outnumbers the number of people on the planet, and Java can play a really important role in the device space." Examples of Java embedded implementations were included in sessions and shown at the demogrounds. For example, the Oracle Russia engineering team demoed an Oracle Java SE Embedded and JavaFX application that commanded a crane.

PDFs of JavaOne sessions are available in the Content Catalog.

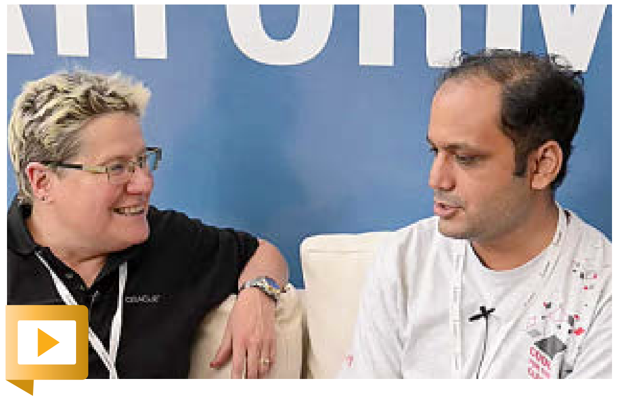
Scenes from Moscow: (top) Oracle's Reza Rahman discusses Java EE 7; (bottom left) Oracle's Nandini Ramani talks tipping points; (bottom right) attendees chill out on giant beanbags.

PHOTOGRAPHS BY VOLTAIRE YAP



Left: The keynote hall in Hyderabad; right: Stephen Chin is all ears at the technical keynote.

JAVAONE LANDS IN HYDERABAD



Tori Wieldt talks with Raj Hedge about Chennai JUG, Adopt a JSR, and more.

Tori Wieldt and Gurpreet Sachdeva chat about the Java garbage collector.

JavaOne India took place May 8–9 in Hyderabad, India. In the opening keynote, Oracle's **Sharat Chander** encouraged attendees to learn new things about Java technology, but just as important, make the effort to meet someone new. "The Java community is open and welcoming, and it will help you to build a network within the community," he said.

Next, Oracle's **Georges Saab** discussed the Java SE and JavaFX shared roadmap. He discussed upcoming features in Java SE 8, and used Project Nashorn as an example of Oracle's commitment to evolve the Java platform. For developers who want to try it, early access builds of JDK 8 are available now. Saab also discussed the growth of embedded devices. "We've already reached the point where there are more devices than people; this is a huge opportunity for developers," he explained. With Java, developers can work from device to data center using the same language and platform.

Oracle's **Anil Gaur** gave an update on Java EE. "When we created Java EE, our primary objective was to build a comprehensive platform to create a wide variety of apps," he said. Java EE has been successful, he added, with millions of developers downloading Java EE around the world.

Stephen Chin kicked off the technical keynote by explaining that he is a developer first, an active Java community member, and a Java evangelist. His goal for JavaOne India is to get developers excited, he said, and he encouraged them to follow their passion.

Next, Oracle's **Jim Weaver** showed the basics of JavaFX 3D, such as including mapping textures to shapes.

Finally, Oracle's **Arun Gupta** walked through the new features of Java EE 7, which provides higher developer productivity and HTML5 support. Community participation was key to creating and testing Java EE 7, he said, with 19 Java user groups adopting various JSRs, testing features, and providing valuable feedback. "It was truly a community effort."

PHOTOGRAPHS BY VOLTAIRE YAP

JAVA USER GROUP PROFILE

FinJUG

The Java User Group Finland (FinJUG) was founded on November 2, 2011, by **Petteri Hietavirta**, who had relocated back to Finland from Scotland, where he had been an active member in Java User Group Scotland. Finding no active Java user group (JUG) in Finland, Hietavirta decided to start FinJUG. A few weeks later, FinJUG held its first "pub meetup" in Helsinki.

More than 130 people are members of FinJUG's Facebook group, and more than 170 people follow @FinJUG on Twitter. The group meets monthly, usually alternating between a pub meetup and a more formal meeting. "Occasionally we organize hackathons and other events," Hietavirta says. "We are quite open topic-wise: anything related to the JVM [Java Virtual Machine] goes."

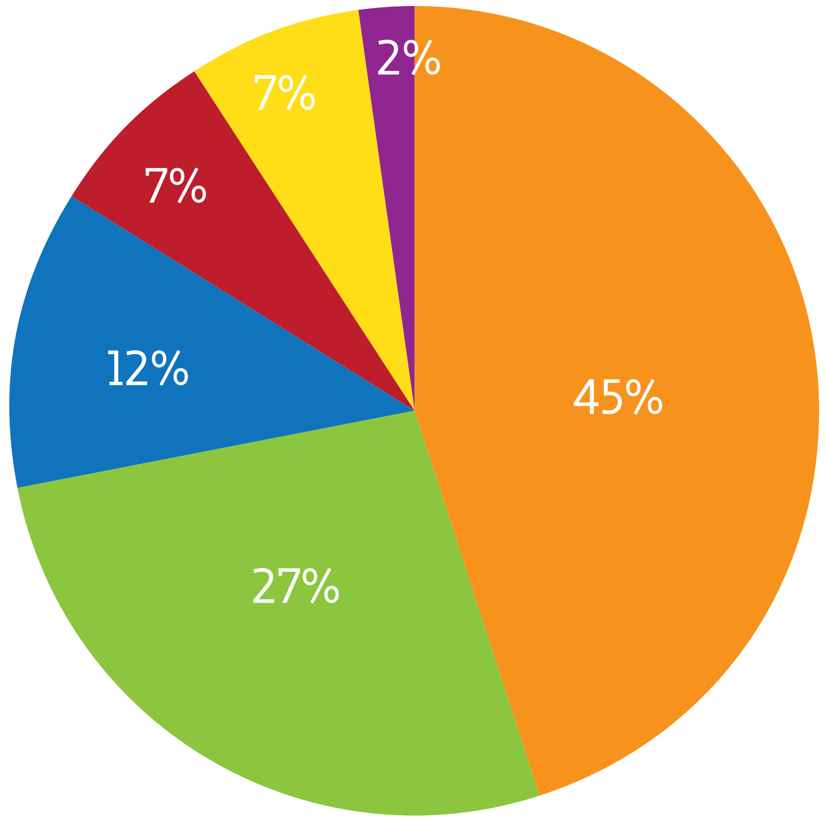
Local businesses have been supportive, providing venues, drinks, and snacks. Manning, O'Reilly, and Zero Turnaround provide giveaways and discounts.

Hietavirta's advice for running a JUG? "In my experience, setting up a JUG is not that difficult. Keep it lightweight, listen to people, and offer opportunities for people to present and organize."

JAVA.NET POLL

Application Performance Is Critical for Success

In a recent Java.net poll, the Java developer community highlighted the importance of performance to the success of the applications they develop. A total of 242 votes were cast during the three-week survey in response to the question "How critical is performance/scalability to the success of the apps you develop?" Here are the final results:



45% Absolutely critical: we spend lots of time and effort tuning performance.

27% It's important: profiling performance is part of our normal development process.

12% We probably should allocate more time to performance issues.

7% Performance doesn't matter that much for our apps.

7% I don't know.

2% Other

JavaOne Russia Duke's Choice Award Winners

At this year's JavaOne Russia, three companies were awarded a 2012 Duke's Choice Award in recognition of their unique Java technology innovations: EPAM Systems, VNIIRA, and JetBrains.

EPAM Systems has developed an all-purpose Java EE adapter that connects Russian banking information systems, facilitating automated data exchange between diverse bank applications.

VNIIRA applied the NetBeans modular platform to develop an air traffic control system that integrates data from primary and secondary surveillance radars to permit tracking of hundreds of flights.

JetBrains was recognized for the new JavaFX support in the latest version of IntelliJ IDEA, for its historical strong support for open source Java, and for its engagement and leadership within the Java community.

ART BY I-HUA CHEN

JAVA CHAMPION PROFILE

ANGELIKA LANGER

Angelika Langer is a German instructor, coach, and author with expertise in advanced C++ and Java programming, including concurrent programming and performance tuning. She is a member of the Java Community Process (JCP) and was named a Java Champion in December 2005.

Java Magazine: Where did you grow up?

Langer: In a steel and coal mining area in western Germany.

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

Langer: I wrote my first program (the sieve of Eratosthenes written in Fortran) at university as a math student, back when programs were delivered as a pile of punch cards for batch execution on a mainframe computer.

Java Magazine: What was your first professional programming job?

Langer: Development of a tool for implementation of telecommunication software. In retrospect, I think this first job kindled my long-lasting interest in software development tools such as code generators, compilers, runtime systems, and so on. I joined a C++ compiler group and got involved in the development of the C++ standard library. Eventually I ended up with Java, where again my key interest is in the language (for example, generics and lambdas), the compiler, the core libraries, and the runtime system.

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

Langer: Hiking, eating, drinking, and cooking—preferably with my spouse.

Java Magazine: What happens on your typical day off?

Langer: I spend it in front of my computer scanning in my negatives, doing digital darkroom work, and arranging images into photo books.

Java Magazine: What "side effects" of your career do you enjoy?

Langer: Traveling and meeting interesting people all over the world. The training business brought me to cities and countries that I would probably never have visited without a client there who called me in for onsite training.

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

Langer: It brought the user groups to my attention. I try to stop by at the local user groups when I'm in town and offer to give a presentation if my schedule allows.

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

Langer: I'm excited about Java 8. Currently, I am putting together explanatory material: a tutorial and reference on lambdas and streams, similar to the material I put together on generics. I'll be running workshops and seminars on lambda/streams, starting this fall with an informal event for programmers who have fun experimenting and exploring the new features playfully. Personally, I look forward to seeing the Java community embrace the new features and I am excited to be part of the effort.

Find more about Angelika Langer at her Website.

EVENTS

JavaOne 2013 *SEPTEMBER 22–26*
SAN FRANCISCO, CALIFORNIA

JavaOne is the meeting place for global experts and decision-makers in the Java community. It is a week of major announcements about the future of the platform, labs and sessions from top speakers, and social gatherings with top figures of the Java world and software industry. Track topics include the core Java platform, Java and security, Java and the cloud, and other mission-critical Java topics. Tracks will also focus on the tools and techniques that help create user experiences that can be delivered through a variety of channels, including personal devices, smartcards, embedded environments, and intelligent equipment.

Qcon São Paulo
AUGUST 29–31
SÃO PAULO, BRAZIL
This enterprise software development conference is dedicated to developers, team leads, architects, and project managers. The conference includes eight tracks and a tutorial day.

JCertif
SEPTEMBER 9–15
BRAZZAVILLE, REPUBLIC OF THE CONGO
JCertif is the biggest IT community event in central Africa. International speakers present talks and labs about Java technologies, Web apps, cloud apps, and more. The event brings together students, developers, and engineers as well as managers, entrepreneurs, senior and midlevel administrators and government employees, and educational leaders.

JavaZone
SEPTEMBER 11–12
OSLO, NORWAY
JavaZone 2013 will be the 12th consecutive JavaZone conference. It offers a combination of technical talks and panels in an informal atmosphere, with an expected attendance of more than 2,000.

PPPJ 2013
SEPTEMBER 11–13
STUTTGART, GERMANY
This conference brings together researchers, teachers, practitioners, and programmers who study or work with the Java platform. This year's focus is the interaction between virtual machine technology and different programming languages.

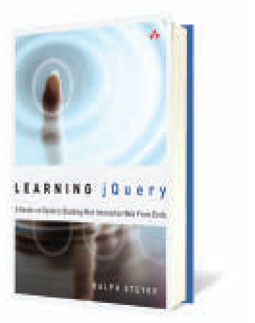
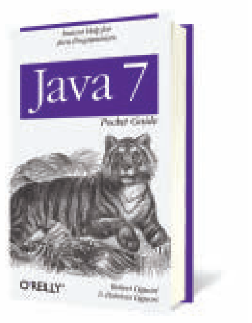
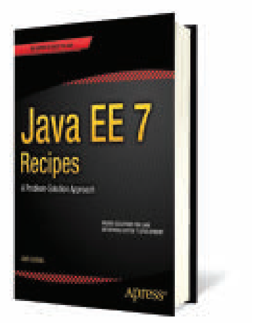
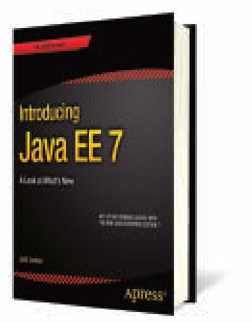
Silicon Valley Code Camp
OCTOBER 5–6
LOS ALTOS HILLS, CALIFORNIA
At this free community event, developers learn from fellow developers. Topics include software development, software branding, and legal issues.

GOTO Aarhus 2013
SEPTEMBER 30–OCTOBER 4
AARHUS, DENMARK
This software development conference is designed for developers, team leads, architects, and project managers. The program includes a three-day conference with 80 presentations and two-day tutorials about architectures, processes, and front-end and back-end software development.

JavaSpotlight Podcast

Listen to the JavaSpotlight podcast for interviews, news, and insight for and from Java developers. Hosted by **Roger Brinkley**, this weekly show includes a rotating panel of all-star Java developers.

JAVA BOOKS

LEARNING JQUERY: A HANDS-ON GUIDE TO BUILDING RICH INTERACTIVE WEB FRONT ENDS
By Ralph Steyer
Addison-Wesley Professional (April 2013)
This book will guide you through using jQuery, jQuery UI, and jQuery Mobile in your own projects. One step at a time, you'll learn how to do everything from adding simple effects through building complete rich internet applications. This code-rich tutorial is designed for every working Web developer. After clearly explaining all the basics, author Ralph Steyer shows how to apply jQuery to create effects, animations, slide shows, lists, drag-and-droppable elements, interactive forms, and much more.

JAVA 7 POCKET GUIDE, SECOND EDITION
By Robert Liguori and Patricia Liguori
O'Reilly (July 2013)
Concise, convenient, and easy to use, *Java 7 Pocket Guide* gives you Java stripped down to its bare essentials—it's a quick reference guide to Java that actually fits in your pocket. Written by Robert and Patricia Liguori, senior software and lead information engineers for Java-based air traffic management and simulation environments, the book gives you the information that you really need to know about Java. This updated edition pays special attention to new areas in Java 7 and Java 8, including lambda expressions.

JAVA EE 7 RECIPES: A PROBLEM-SOLUTION APPROACH
By Josh Juneau
Apress (June 2013)
Java EE 7 Recipes takes an example-based approach in showing how to program enterprise Java applications for many different scenarios. Whether you are working on a small-business Web application or an enterprise database application, *Java EE 7 Recipes* provides effective and proven solutions to accomplish just about any task that you may encounter. The solutions are built using the most-current Java enterprise technologies, including Enterprise JavaBeans (EJB) 3.2, JavaServer Faces (JSF) 2.2, Expression Language (EL) 3.0, Servlet 3.1, and JavaFX 2.2.

INTRODUCING JAVA EE 7: A LOOK AT WHAT'S NEW
By Josh Juneau
Apress (June 2013)
This book guides you through the new features and enhancements in Java EE 7. Readers will not have to wade through introductory material or information about features that have been part of the platform for years. Instead, they can pick up this book to learn about those features that have changed or have been added for the Java EE 7 release. This reference helps you move forward from Java EE 6 to the new Java EE 7 platform quickly and easily.

GET READY FOR JAVAONE

The conference returns to San Francisco, September 22–26.

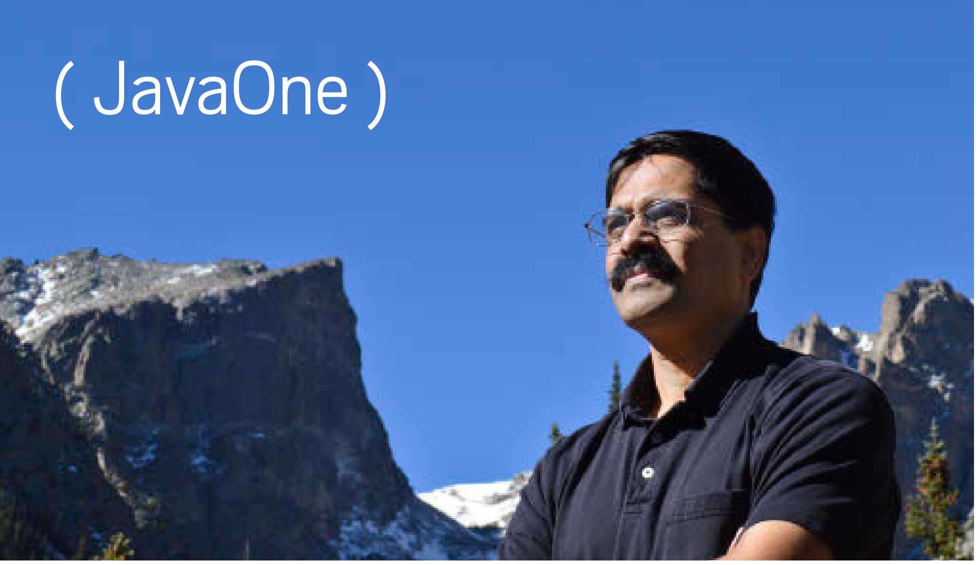
By Stephen Chin and Kevin Farnham

At the JavaOne 2012 Community Keynote, Oracle Java Technology Evangelist Stephen Chin (@steveonjava) was announced as the newly appointed JavaOne community chairperson. Chin is an author and speaker, a JavaFX evangelist, a Java embedded technology enthusiast, a JavaOne Rock Star, and a Java Champion.

Among Chin's roles in his new position is chairing the committee that reviews JavaOne session proposals. For JavaOne 2013, a new "rolling admission" process was used by the selection committee, whereby session acceptances were sent out in multiple batches. The first group of JavaOne invitees was announced in Chin's April 10 blog post, "Congrats to the First JavaOne Invitees!"

This new rolling admission system enabled *Java Magazine* to interview four experts in diverse Java technologies (all of whom will be presenting at JavaOne 2013) about what they're working on, the technologies that excite them, and what they look forward to at this year's conference.

ART BY I-HUA CHEN



MEET A SPEAKER

VENKAT SUBRAMANIAM

Dr. Venkat Subramaniam is an award-winning author, founder of Agile Developer, and an adjunct faculty member at the University of Houston. He has trained and mentored thousands of software developers across the globe. Follow him on Twitter (@venkat_s).

Java Magazine: How has being a Java Champion affected your professional life thus far?
Subramaniam: The most significant impact has been really getting to know some of the wonderful professionals around the world who are members of this program.
Java Magazine: You recently wrote the book *Functional Programming in Java: Harnessing the Power of Java 8 Lambda Expressions*. What makes you excited about lambda expressions?
Subramaniam: I would summarize this as power to programmers. Functional programming offers a concise, declarative style of programming. With languages on the JVM [Java Virtual Machine] that already provide this, Java 8 offers Java programmers the ability to be very expressive.
Java Magazine: Tell us about one of the sessions you'll be presenting at JavaOne.
Subramaniam: I'm looking forward to talking about JVM languages and some of the wonderful things we can do with languages such as Groovy, Scala, Clojure, and definitely Java 8 as well.
Java Magazine: Education is a major focus of your professional career. To get the most out of the conference, what advice would you give to developers attending JavaOne?
Subramaniam: Attend talks about technologies that are different from the ones you are using right now. When we listen to a different viewpoint, we pick up new ideas. It doesn't mean we have to drop what we're doing. But it helps us to rethink the way we might be programming in the mainstream language we're using.
Java Magazine: What are you looking forward to at JavaOne?
Subramaniam: Learning from fellow developers—the hallway conversations, the dinner conversations, learning from what other people are doing, what they are curious about, what they are investigating.

DID YOU KNOW?

SAN FRANCISCO WAS ORIGINALLY CALLED YERBA BUENA, MEANING "GOOD HERB" IN SPANISH, IN REFERENCE TO THE WILD MINT GROWING NEARBY.

Get Your Fix

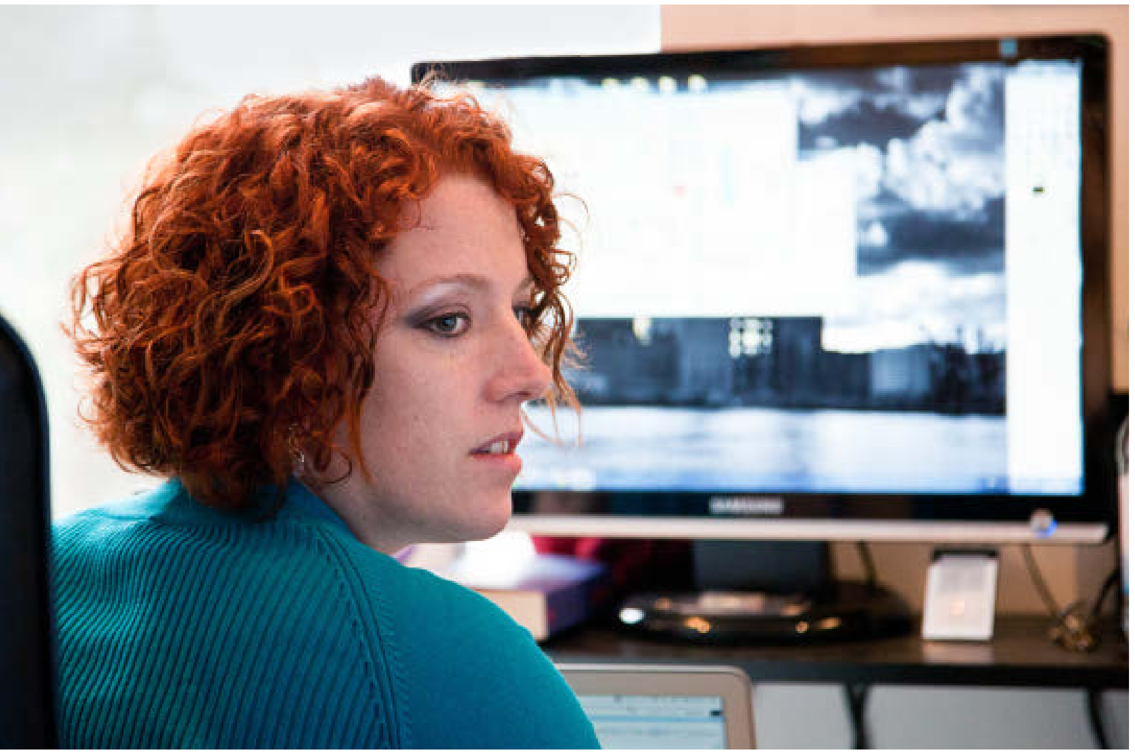
Get **COFFEE** without leaving the vicinity of the conference. This list of java joints is arranged by walking time from JavaOne at the Hilton Union Square.

Starbucks at The Hilton Union Square
333 O'Farrell Street

Taylor Street Coffee Shop
375 Taylor Street
2-minute walk
Although it offers only plain coffee and no specialty drinks, it also serves a hearty breakfast.

Café Encore
488 Post Street
3-minute walk

Barbary Coast
55 Cyril Magnin Street
4-minute walk



PHOTOGRAPH BY JOHN BLYTHE

MEET A SPEAKER

TRISHA GEE

Trisha Gee (@trisha_gee) is an active leader of the London Java Community. She works for 10gen and is passionate about helping increase developer productivity.

Java Magazine: On your blog, you say your goal is "only to change the world." What are you working on right now to make that happen with respect to Java and JVM [Java Virtual Machine]–related technology?

Gee: I'm working on a new Java driver for MongoDB. You can make people's lives easier by tiny things like saving them seconds or minutes. My aim is to make other developers' lives as easy as possible.

Java Magazine: Tell us about one of the sessions you'll be presenting at JavaOne.

Gee: I'm presenting a session called "Design as a Process, not an Offset." It documents some of the trials and tribulations we went through when we were designing the new Java driver for MongoDB. You're constantly going into the process of design when you're developing. And we don't necessarily realize we're doing that, so I wanted to call that out.

Java Magazine: Java user groups [JUGs] and JavaOne have formed a symbiotic relationship. What benefits does JavaOne provide to JUGs?

Gee: The thing that is awesome about JavaOne is meeting people from other user groups. You go to San Francisco and you meet people from the US but also from Africa and Brazil. There's nothing really quite like actually meeting someone face-to-face and having a chat about some of the problems we're about to face.

Java Magazine: Aside from your own sessions, what are you looking forward to at JavaOne?

Gee: I get the most value out of JavaOne from the people I meet. There are amazing speakers, and you get a chance to meet them and get firsthand knowledge of something that you normally only read about in books or blogs, and that's really awesome. I don't think you can get that from many other conferences. I think JavaOne is the ultimate global conference for anyone who's working on Java or JVM languages. It brings everyone together in one place.

DID YOU KNOW?

THE TRANSAMERICA PYRAMID IS THE TALLEST BUILDING IN SAN FRANCISCO.

Get Your Fix

Cable Car Coffee Company
900 Market Street
4-minute walk

Café Madeleine
43 O'Farrell Street
5-minute walk
This café is known for its mochas, which are made with real ganache.

Sugar Café
679 Sutter Street
7-minute walk
Try the iced caramel macchiato on a warm day.

Blue Bottle Coffee
66 Mint Plaza
7-minute walk

The Coffee Bean & Tea Leaf
773 Market Street
7-minute walk
Try the lightest roast black coffee.

—Curran Mahowald

MEET A SPEAKER

GERRIT GRUNWALD

Gerrit Grunwald (@hansolo_) is a Java user group (JUG) leader and a JavaFX community leader from Münster, Germany. Recently, he has been actively engaged with Java in embedded devices. Grunwald blogs at Harmonic Code.

Java Magazine: Your interests include an unusual pairing of technologies: JavaFX and HTML5 on the one hand, and embedded devices on the other. What are you currently working on with these technologies?

Grunwald: I'm very interested in JavaFX and HTML5. Also, I studied physics, and that's the reason behind my interest in embedded devices. When I heard last year at JavaOne that JavaFX and Java will come to the Raspberry Pi, I was really thrilled by the idea that I can use my favorite technology on these devices. At the moment, I'm trying to figure out what you can do with this mix of technologies. It's really interesting to figure out the possibilities of having Java and JavaFX on embedded devices.

Java Magazine: Tell us about a session you'll be presenting at JavaOne.

Grunwald: One session is about JavaFX. Many people ask me, "How do you create all of that graphical stuff for JavaFX controls and all that?" And I have to explain it to them. So, I thought it might be a good idea to create a session that explains a lot of the stuff that I'm doing and how I do it.

Java Magazine: What's the greatest benefit in attending JavaOne?

Grunwald: The community, that's really it. And going to JavaOne is really meeting all the people. You have the chance to talk to all of these different experts face-to-face, not on e-mail or Twitter or something. You can go to them and ask them questions. You can get great ideas and great discussion. That's what I like most about going to JavaOne.

Java Magazine: Aside from your own sessions, what are you looking forward to at JavaOne?

Grunwald: Meeting with JavaFX guys from Oracle, and with the people who are working with JavaFX in general. The JavaFX community, the desktop community, meeting these people: it's like meeting with your family.

DID YOU KNOW?

DENIM JEANS WERE INVENTED IN SAN FRANCISCO FOR GOLD MINERS.

Quench Your Thirst

Check out the eclectic **BAR SCENE** in San Francisco.

The Buena Vista Café
2765 Hyde Street
Try the famous Irish coffee, which was purportedly invented here.

Bourbon and Branch
501 Jones Street
Experience a speakeasy from the Prohibition Era (make reservations to get the password).

Martuni's
4 Valencia Street
Sip a martini and sing along with the piano player.

Jasper's Corner Tap
401 Taylor Street
Want a beer? They have 18 on tap (plus food and cocktails).

PHOTOGRAPH BY helenecanada/GETTY IMAGES

MEET A SPEAKER

ANTON (TONI) EPPLE

Toni Epple (@monacotoni) is a JavaFX, Swing, and NetBeans trainer and consultant. He is coleader of the JavaTools community at Java.net and a member of the NetBeans Dream Team and the NetBeans Governance Board.

Java Magazine: What are you working on right now that you believe will ultimately substantially benefit Java/JVM [Java Virtual Machine] developers?

Epple: I'll be teaching JavaFX at a school in Munich, helping the pupils become the next generation of Java community members. And I'm organizing a conference called JayDay.

Java Magazine: Tell us about one of the sessions you'll be presenting at JavaOne.

Epple: "Angry Nerds Part 2" is a tutorial session about creating games using JavaFX.

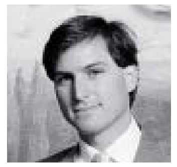
Java Magazine: As a professional trainer, you know that companies are interested in increasing the skills of their development teams. What's the benefit for companies in sending key members of their development team to JavaOne?

Epple: I see JavaOne as the most important Java conference because it's the one organized by the developers of Java. If you're evaluating a new technology and you send your developers to JavaOne, they have a very good chance to talk to the creators of that technology. Also, at JavaOne you can recharge your developers with new knowledge and new energy. That should be worth the ticket.

Java Magazine: Aside from your own sessions, what are you looking forward to at JavaOne?

Epple: I'm looking forward to seeing a lot of good presentations, especially from the Oracle people. JavaOne is the best place to get the latest information about new Java developments, and get input from its creators.

PHOTOGRAPHS BY EROL GURIAN AND GETTY IMAGES



FAST FACT

WHAT DO STEVE JOBS, CLINT EASTWOOD, AND ROBERT FROST HAVE IN COMMON? *THEY WERE ALL BORN IN SAN FRANCISCO.*

Quench Your Thirst

Gitane
6 Claude Lane
1930s cabaret, 1970s disco, and Spanish cuisine meet

Smuggler's Cove
650 Gough Street
Lots of rum (400 types)

Fat Angel
1740 O'Farrell Street
Unique beers, wine, and tasty morsels

Maven
598 Haight Street
Creative cocktails paired with small plates

Rye
688 Geary Street
Unique cocktails plus late-night eats

Hi Tops
2247 Market Street
Castro district sports bar

—Curran Mahowald

GEEKS, TO YOUR BIKES!

The Geek Bike Ride takes place on Saturday, September 21, at 9:30 a.m. Enjoy the best vistas of the Bay Area by riding across the Golden Gate Bridge. Riders will meet at Blazing Saddles in Fisherman's Wharf and ride across the bridge and down into Sausalito, and then take a ferry back to the city. This beginner/intermediate ride is roughly 8 miles and takes 1.5 hours. Bike rentals are US$30–US$40, with a 10 percent discount if you reserve online. The ferry ride is US$10.50. Wear your Java Geek bike jersey if you have one. All geeks, friends, and family are invited. Look for #geekbikeride on Twitter for updates.

PHOTOGRAPHS BY YOSHIO TERADA

JavaOne Tracks

Track topics this year range from the stronger-than-ever **core Java platform** to in-depth and timely explorations of **Java and security**, **Java and the cloud**, and client and embedded development with **JavaFX**. Tracks will also focus on edge computing with Java in **smartcards, embedded environments, and intelligent equipment**; **emerging languages on the Java Virtual Machine**; **development tools and techniques**; and **Java EE**.

Get Touristy

Have the quintessential **Fog City experience** while you are in town. Tourists love these activities—with good reason.

Visit Alcatraz Island. Take a ferry over to this military-post-turned-prison in the San Francisco Bay. The island, aka "The Rock," is crawling with history and wildlife.

Visit Coit Tower. Perched atop Telegraph Hill in North Beach, Coit Tower was built in the art deco style in 1933 to honor the city's volunteer firefighters.

Cross the Golden Gate Bridge. Walk, run, or ride a bicycle across the Golden Gate Bridge to breathe in the crisp ocean air while getting some spectacular views.

Taste wine in Sonoma, Napa Valley, or Alexander Valley. Take advantage of the proximity to California wine country.

Take a helicopter tour. Get a new perspective on San Francisco.

—Curran Mahowald

PHOTOGRAPHS BY VICTOR BRODSKY AND NOAH CLAYTON/GETTY IMAGES

JAVAONE SCHEDULE HIGHLIGHTS

Saturday, September 21	
7:00 a.m.	Registration Opens, Moscone Center
9:30 a.m.	Geek Bike Ride, Fisherman's Wharf
Sunday, September 22	
7:30 a.m.	Registration at Moscone Center, Hilton San Francisco Union Square, Masonic Center (3:00 p.m.–7:00 p.m. only)
8:00 a.m.	Java University
9:00 a.m.	User Group Forum, Moscone West
4:00 p.m.	Java Strategy and Technical Keynotes, Masonic Auditorium
7:00 p.m.	Taylor Street Open House
8:00 p.m.	GlassFish Party, Thirsty Bear
Monday, September 23	
8:30 a.m.	Sessions, Hands-on Labs and Tutorials, and Birds-of-a-Feather Sessions Begin and Continue Through Wednesday
9:30 a.m.	Exhibit Hall Opens
Wednesday, September 25	
3:00 p.m.	Exhibit Hall Closes
7:30 p.m.	Appreciation Event, Treasure Island
Thursday, September 26	
9:00 a.m.	Java Community Keynote, Hilton San Francisco Union Square
4:30 p.m.	It's a Wrap, Yerba Buena Gardens

Left to right: Twitter's Chris Lambert, engineering manager; Ben Hindman, software engineer, runtime systems; and Robert Benson, senior director of software engineering, discuss the company's cloud infrastructure.

#performance

Twitter migrates core infrastructure to the JVM and supports more than 400 million Tweets per day. **BY DAVID BAUM AND ED BAUM**

PHOTOGRAPHY BY BOB ADLER

Twitter's 2006 launch was a long time ago in internet years—and the real-time information network has been on an upward trajectory ever since. By 2009, journalists were reporting that

SNAPSHOT

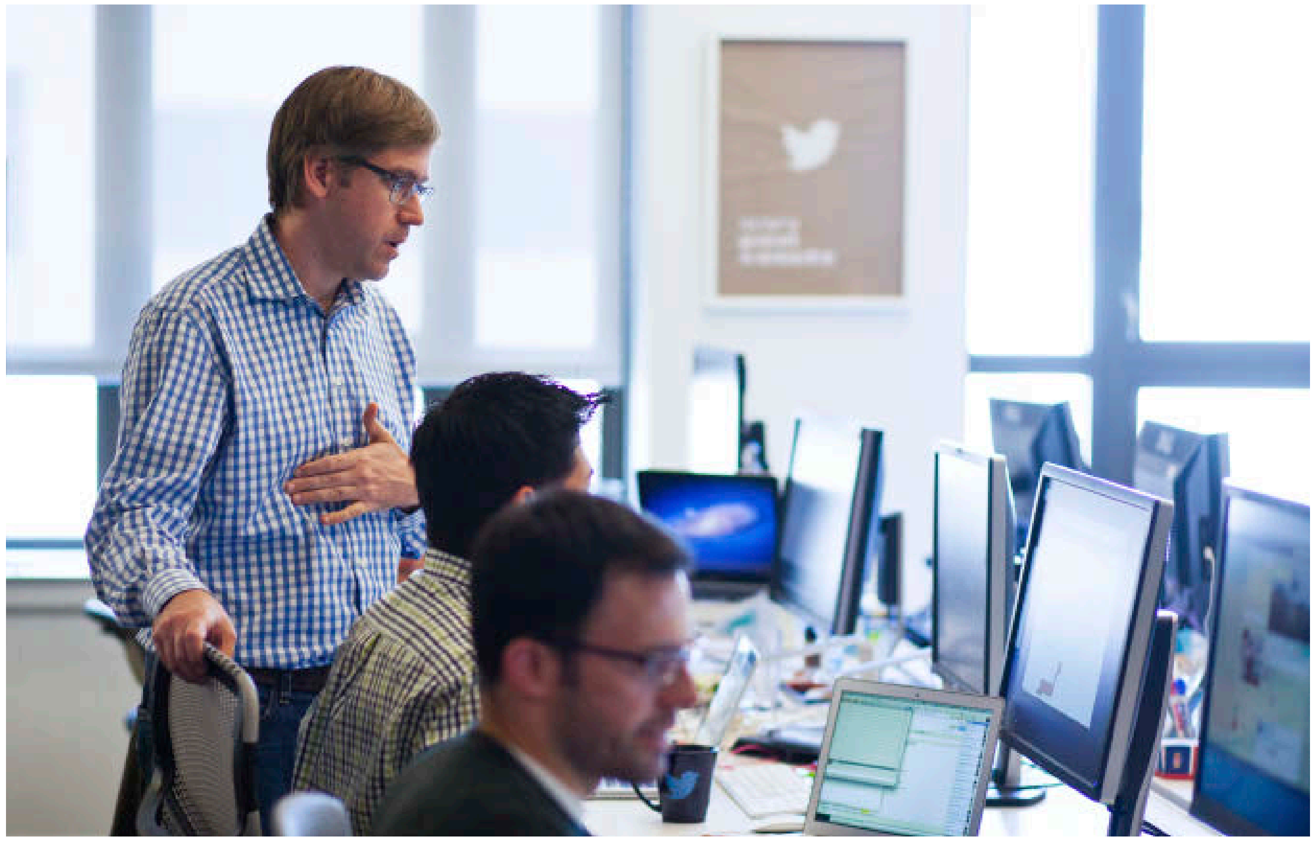
TWITTER
twitter.com

Headquarters:
San Francisco, California

Industry:
Media and technology

Employees:
1,300

Java technologies used:
Java SE 1.7, JDK 7



Benson gets a project update from Tung Vo, senior manager of software engineering.

sheer numbers alone couldn't possibly describe Twitter's social impact. Like the internet itself, the larger Twitter's user base grew, the more useful the service became. Today Twitter is in the vanguard of the social Web, an emerging cultural network in which static content from a relatively small group of publishers is being replaced by free-form, dynamic interaction between millions of participants. And while analysts debate whether Web 2.0 is truly a new paradigm or merely the collaborative medium the internet was originally envisioned to be, Twitter's more than 200 million active users aren't wasting their 140 characters arguing about internet semantics. They're simply tweeting—to the tune of more than 400 million Tweets per day—as they discuss every topic imaginable, from casual chatter to world-changing social and political issues.

Thanks to Java, Twitter facilitates those discussions more competently than ever.

"Performance is one of the most important products that any service can deliver to its customers," says Robert Benson, senior director of software engineering at Twitter. "End users want Twitter to be fast so they can get real-time information. Reliability and performance are huge goals for us, and that's why part of our core strategy involves moving to the Java Virtual Machine [JVM] runtime environment. Twitter no longer has the performance issues it previously had, and that's in large part due to our moving to the JVM."

TWITTER BY THE NUMBERS
Average number of search queries per day: 1.6 billion

BEACHING THE FAIL WHALE

Twitter is one of the top 10 most visited sites on the internet. Unregistered users can read Tweets, and registered users can post them via the Web, SMS, or apps for mobile devices. Part of what makes Twitter attractive is its user-friendly functionality. Hashtags, trending topics, following, @replying, and retweeting all contribute to its ease of use and popularity.

Engineers at Twitter admit that while the company has always cared deeply about the quality of its service, delivering on that has been challenging in the face of such explosive growth.

Most longtime Twitter users are familiar with the fail whale error message. The whale illustration, created by Chinese-Australian artist Yiying Lu, pops up to inform users that Twitter is over capac-

Benson and Hindman chat in one of Twitter's casual conference booths.

ity and urges them to try again later. But service outages on Twitter have been noticeably less frequent since late 2010. That's no coincidence. Benson says that Twitter's engineers have been doing a lot of thinking about Twitter's architecture and the challenge of handling so many requests every second. Years of constantly refining their approach have brought them to a solution that uses the JVM to build systems that can handle that load easily by scaling horizontally.

"We don't want to depend upon machines getting taller and taller or the resources in those individual boxes increasing," Benson says. "We want computers that can handle requests in parallel. The JVM is a managed language runtime that can deal with concurrency in a very efficient way. It can handle these types of workloads. A great deal of our data center was dedicated to handling the API traffic of our customers. That can now be managed with far fewer machines on the JVM while delivering huge boosts in performance."

The Twitter team has moved many of the company's most critical systems to a set of services written in Java and Scala running on the JVM. The service is now a worldwide presence that can capably handle sustained peak levels during major events like the Super Bowl and the US presidential election without an appearance from the fail whale. Users enjoy a very fast system that enables them to get information within seconds about events taking place all over the world.

Benson says the migration to the JVM not only delivers performance wins; it also provides something he likes to call observability. "Running a service of this scale, things go wrong all the time, either because of runtime issues or because software is being deployed every hour," he says. "We want to be sure we understand why those failures happen. With the JVM, it is a lot easier for us to examine those events in a robust way than it was with other runtimes we have used in the past."

Finally, one of the critical factors in moving to the JVM was the OpenJDK open source project. "As the guy who needs to think strategically about how my organization can work, not only inside the building but also externally, the open source nature of the JVM is very important to me because we can see the source on which we're building the core infrastructure," Benson says. "We hire engineers to work with that codebase and community to improve the runtime so that we can build faster and more-predictable services on top of the JVM. These are the kinds of things that made the choice of migrating to the JVM easy."

TWITTER BY THE NUMBERS
Monthly active Twitter users:
More than 200 million

GIVING BACK
"With a cloud infrastructure, we have multiple JVMs that run side by side in a single box. We are actively investigating ways to… minimize performance blips due to the colocation of workloads. Our goal is to contribute back to the OpenJDK community so that we help push that platform forward."

EQUAL TIME FOR THE ETHEREAL AND THE MUNDANE

The traffic on Twitter runs the gamut from logistical instructions ("September 17. Zuccotti Park. Bring tent. #OccupyWallStreet") to simple details of people's daily lives ("Corn dogs for lunch again. #Winning"). But no matter what people tweet about, they want Twitter to perform the same way, every time they use it.

Benson says the JVM plays an important role in refining the predictability of the service. "Because the JVM is a managed runtime, our developers can work more quickly," he explains. "They don't need to worry about manually managing memory. But the trade-off is, we need the JVM to efficiently manage garbage, which are objects that the process doesn't need anymore."

Predictability is difficult to achieve in the garbage collection process. Benson and his team are investing engineering resources in endeavoring to fine-tune the JVM to make it a more predictable garbage collector. "This is critical in a high-throughput, low-latency system like ours," he continues. "We are working with people in the JVM community to improve the garbage collection strategies and heuristics."

Benson's team is also helping to improve the JVM as Twitter moves toward a cloud architecture where there are no dedicated boxes for anything and processes can flow around a data center, automatically rescheduled in the wake of failures. "With a cloud infrastructure, we have multiple JVMs that run side by side in a single box," he notes. "We are actively investigating ways to improve how that works to minimize performance blips due to the colocation of workloads. Our goal is to contribute back to the OpenJDK community so that we help push that platform forward."

FOLLOWING THE FLIGHT OF A TWEET

To help explain the importance of the JVM within the Twitter infrastructure, here is a high-level overview of the primary ways in which the JVM affects an average Tweet as it is set loose into the Twittersphere:

1. A popular fashion pundit sees a Best Actress nominee emerge from her limousine on the eve-

Benson discusses hiring with recruiter Christian Bogeberg (left) and Hindman in Twitter's Commons, where three meals a day are served to employees, free of charge.

TWITTER BY THE NUMBERS

Average number of Tweets sent per day: more than 400 million

ning of the Academy Awards. She types her impressions into a Twitter app on her smartphone: "Glamorous Jennifer Lawrence approaches red carpet in white Dior gown evoking old Hollywood. #OscarsFashion2013." Her Tweet is geotagged with geographical metadata noting the location.

2. After she hits the Tweet button, her message is sent to Twitter's front-end routing and load-balancing tier, which is written in Scala and runs on the JVM. The front-end system communicates with Twitter's business tier, which also runs Scala, to authenticate the user and verify that her incoming Tweet conforms to the site's business rules.
3. The Tweet enters an asynchronous pipeline that will end up delivering it to the caching and storage tiers. Once the Tweet enters this pipeline, the system sends back a reply informing her that her Tweet was successfully posted. In the background, the Tweet is then delivered to all the users who follow this popular fashion authority.
4. Next the Tweet enters Twitter's relevance pipeline, which determines what it is about and who will be interested in it. Relevance is based not only on followers, but also on information within the Tweet: who the sender is, where it was sent from (geotag), and the hashtag that categorizes Tweets by keywords.
5. A Java program in this pipeline recognizes that the hashtag #OscarsFashion2013 is being used on a large number of Tweets and adds that hashtag to Twitter's trending topics. Instantly, Twitter users around the world see #OscarsFashion2013 as a topic of interest and can send or receive Tweets with that hashtag.
6. When another fashion-minded user sees the trending #OscarsFashion2013 hashtag, he or she can click on it—which results in a Scala program retrieving the Tweet from the cache or storage tiers.

GETTING THE MOST FROM JAVA

Benson believes his team is pushing the JVM in ways that it has rarely been pushed elsewhere. "Our latency requirements, the predictability of the garbage collection, the number of languages we run on the JVM, and much more make Twitter an exciting place to work," he says. "The JVM gives us the flexibility to construct new and better products."

As talented Java and Scala developers work on Twitter's data systems, analytic systems, client infrastructure, and multiple other projects, they do more than enrich the Twitter experience for end users. They also help enhance the Java platform. That's something most service companies don't have the knowledge or wherewithal to do, so it's no surprise that Twitter attracts some of the brightest software engineers in the industry.

"There are many reasons the JVM is a great fit for us," Benson summarizes. "We benefit from concurrency, ability to run multiple languages in the same process, and observability of the JVM. Our operations are more cost-effective because we can do more with less in our data centers. Most importantly, we can deliver reliable service to our customers. The JVM gives us many tools we need to experiment, iterate, and quickly deliver what the world wants." </article>

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

WIND POWERED. DATA POWERED.

ORACLE TEAM USA relies on a wireless Java system for real-time data to improve performance on the racecourse.

BY PHILIP J. GILL

PHOTOGRAPHY: © ORACLE TEAM USA / PHOTO: GUILAIN GRENIER

SNAPSHOT

ORACLE TEAM USA
oracle-team-usa.americascup.com

Headquarters:
San Francisco, California

On-board crew members:
11

Java technology used:
Java SE 1.7

The America's Cup, the world's oldest and most prestigious sailing yacht competition, is essentially a new and different race each and every time it takes place. For this year's challenge, the 34th America's Cup, there are new rules and a new location, San Francisco Bay.

There's also a new class of sailing craft designed and built exclusively for this race. "The new rules call for a 72-foot catamaran," explains Gilberto Nobili, a member of the ORACLE TEAM USA crew better known by his nickname, "Gillo." "One of the main characteristics of this new boat is that it doesn't have a soft sail; it has a main sail that is a fixed wing, like the hard wing of a plane."

"The other main characteristic is the 72-foot box rule," Nobili continues, noting that the boat is called an AC72. "That's the fixed measurement of the boat platform that also has a fixed weight, and you have to stay inside the limits of that box. But inside those limits, you can do whatever you want to make the boat faster."

In designing and building its AC72-class boat, *ORACLE TEAM USA 17*, the team—which won the 33rd America's Cup in 2010 to become this edition's defender—has made extensive use of a custom-built Java data collection and information delivery system that Nobili writes, maintains, and refines. In addition to these duties, Nobili is also one of the boat's *grinders*—crew members who manipulate the boat's sails using winches to pull them in, let them out, haul them up, or bring them down. It is considered the most physically demanding post on the crew.

THE RACE FOR THE CUP
This year's America's Cup Finals begin September 7 on San Francisco Bay.

"The Java system has two main purposes," says Nobili. "One is for designing and developing the boat." He explains that the other purpose is to collect and use real-time data for sailing the boat, "so while we're sailing, we're using numbers to get better performance or to go around the racecourse."

JAVA ON BOARD

For the past several months, says Nobili, the ORACLE TEAM USA crew has been using a test boat to prepare for the race and to help refine the design of the new craft that recently launched in advance of the America's Cup Finals in September. Made of carbon fiber to minimize its weight, the test boat's hull is honeycombed with special sensors that collect data.

"We collect about 3,000 variables 10 times a second, every single second we are out sailing on our test boat," says Nobili. "We use that data to study and improve the boat, to check the loads, and to improve the design of the boat we built for the race."

On race days in September, the same Java data collection system will also provide critical information to the crew during operations that will help them achieve maximum speed and, hopefully, sail to victory.

This year's America's Cup is the 34th challenge in the race's 162-year history, and one thing that hasn't changed in all those years is that the boat must sail under human power—no engines are allowed. "You cannot ever store energy on the boat, and you cannot use an engine to move or to stop," says Nobili. "So on board, you need a few big guys who have the physical strength to manually adjust the sails and, basically, fight the wind. These guys use a thing called a grinder [from which Nobili's crew member role gets its name], which is a pedestal with two handles, like a bike, and we use our hands to manipulate the grinder, and that moves the sails."

"We have computers on board to correlate and analyze data, cross-processing in real time," Nobili continues. "But you cannot adjust the setting of the boat by computer. Every adjustment has to be done by a human, and you can read the data from the computers and then have humans make adjustments based on reading that data."

To get the data, the boat's computers collect information from sev-

Left: Gilberto Nobili carries suitcases that hold data-rich devices for each crew member. Right: ORACLE TEAM USA practices in San Francisco Bay.

THE ROLE OF ENGINES

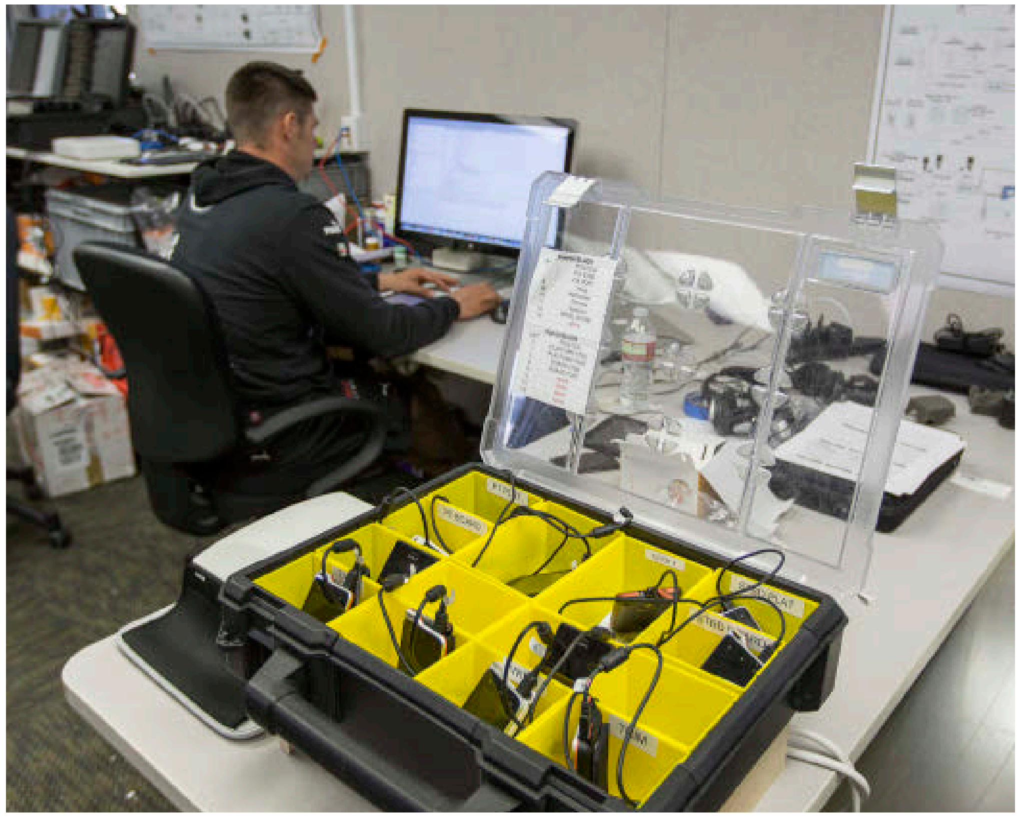
The 33rd America's Cup was the first and only time engines were used in the race. The engines were used to run the winches on the boat. Per America's Cup rules, the boat itself still had to be propelled by sails.

eral sources. "The boat is made of carbon, and inside the carbon we have fiber optics that are monitoring the structure and performance of the boat," says Nobili. "Plus we have GPS, weather information services, and sensors that monitor our high-pressure hydraulics system. We have a lot of sensors to improve the reading of the wind and wind direction, because of course you read the wind when you are sailing—that's an important component of the speed of the boat."

DATA INTO INFORMATION

Turning all this data into information that crew members can use to work the boat is where Nobili's unique combination of skills comes into play. His two passions are computers—particularly coding—and sailing. He studied electronic engineering at a university near Parma, Italy, leaving before graduation to join an Italian sailing team that unsuccessfully challenged the Royal New Zealand Yacht Squadron of Auckland, New Zealand, for the America's Cup in 2000. This year's America's Cup will be his fourth cup race, and his second sailing for ORACLE TEAM USA.

Because of his experience as a grinder, Nobili knows the kind of



Top: Portable devices, which are strapped to crew members' wrists, provide real-time information during practices and races. Bottom: Crew members run into position.

The America's Cup

In 1851, when the schooner *America* set sail from New York, New York, all its backers and crew hoped for was to make back the money it had cost to build the ship. They could not have had any idea that they were launching the world's oldest and most prestigious sailing yacht competition, the America's Cup.

Upon reaching Europe, *America* accepted a challenge from the UK's Royal Yacht Squadron to circumvent the Isle of Wight, the largest British island in the English Channel. *America* won the regatta by 18 minutes. Although the ship itself changed hands many times thereafter, in 1857 the winners donated the silver trophy cup to the New York Yacht Club (NYYC) through a special Deed of Gift that made it available in perpetuity to promote friendly sailing competitions between nations.

Technically, an America's Cup match race is between yacht clubs—which, in turn, field teams backed by wealthy individuals and corporations. The winner one year becomes the defender and gets to write the rules for the next race, including location, course, type of boat, and so on. Ultimately there is only one official challenger, but any number of contenders can vie for the challenger spot if they meet the terms the defender sets forth. When multiple teams qualify, they face off in the Louis Vuitton Cup, a series of races to be held this year in July and August.

Through 25 challenges, the America's Cup remained at the NYYC until 1983, when the Royal Perth Yacht Club of Perth, Australia, became the first successful contender from outside the US. In 1988 the San Diego Yacht Club of San Diego, California, reclaimed the cup for an American team, only to be successfully challenged in 2000 by the Royal New Zealand Yacht Squadron (RNZYS) of Auckland, New Zealand, represented by Emirates Team New Zealand. In 2003 the Société Nautique de Genève of Geneva, Switzerland, mounted a successful challenge, and in 2010 challenger ORACLE TEAM USA—then known as BMW ORACLE Racing—won the competition in Valencia, Spain, and brought the cup home to the United States and the Golden Gate Yacht Club (GGYC) in San Francisco, California.

This year, the official challenger is the Royal Swedish Yacht Club (KSSS) of Stockholm, Sweden, represented by Artemis Racing. The KSSS will first have to compete with the Emirates Team New Zealand and Circolo della Vela Sicilia Yacht Club (represented by Luna Rossa Challenge) teams in the Louis Vuitton Cup. The winner will challenge the GGYC and ORACLE TEAM USA.

information his fellow crew members need, as well as what format best suits their needs. "The data is transmitted wirelessly to small portable Android devices that are a cross between wristwatches and tablets. They are strapped to crew members' wrists," Nobili explains. "The information is delivered as text and numbers; when the boat is moving, there isn't time to read graphics."

On a typical day, the crew hits the gym for a few hours first thing in the morning, and then they're off sailing a test craft for practice. At the end of the day, while his teammates are working on the boat, Nobili sits in the office and refines the Java code that makes up the critical real-time information system, based on feedback from the crew that day.

"It's important that this part of the code is right there, especially the human interface side, the one I am in charge of," says Nobili. "It's all done right there in Java. I'm using a tablet and a heads-up display to give the guys on board the information they need while they are sailing. This is really my job, because I'm sailing—so I know what they need and I try to translate that and make the technology to do that."

In 2010, when ORACLE TEAM USA—then known as BMW ORACLE Racing—won the America's Cup for the first time, only half of the team members wore the wristwatch-like devices, says Nobili. This year, the whole team will have them, and Nobili is quite sure ORACLE TEAM USA won't be the only team using such systems.

Nobili had his choice of programming languages to write the real-time system; the boat's server-side applications, for example, are written in C++. He chose Java, however, for its "write once, run everywhere" portability. "I knew the system needed to be able to run on multiple devices and multiple operating systems," he explains. "Java provides the best way to do that." </article>

Watch ORACLE TEAM USA in action on San Francisco Bay. Crew member Tom Slingsby describes some techniques that the team is testing.

Philip J. Gill is a San Diego, California–based freelance writer and editor who has followed Java technology for 20 years.

JAVA PERFORMANCE TUNING

A conversation with **Kirk Pepperdine** about the ever-changing challenges of Java performance tuning **BY JANICE J. HEISS**

PHOTOGRAPHY BY ARPAD KURUCZ/GETTY IMAGES

Few people, if any, know more about Java performance tuning than Java Champion Kirk Pepperdine, currently a principal consultant at Kodewerk, a company that offers performance-related services, training, and custom tooling.

Pepperdine, a leading consultant who has shared his insight on performance tuning at conferences throughout the world, is highly regarded for his workshops and articles. In a world of rapid changes in both hardware and software, his voice is well worth hearing.

We met with him to discuss the significance of the cloud and increased computing capacity with multicore processors, and how to cope with performance problems in Java enterprise computing.

Kirk Pepperdine takes a stroll near Deák Ferenc tér in Budapest, Hungary.

Java Magazine: Which Java performance tuning tips that were valid five years ago are no longer valid?
Pepperdine: The list is huge. The technology keeps changing, and as it changes, we have to re-evaluate what works and what doesn't work. For example, the JVM [Java Virtual Machine] is currently going through a minirevolution in terms of what it can do for you. It has added a new garbage collector that's just starting to come online. And the other garbage collectors are maturing. The adaptive size policy has been completely rewritten for JDK 8, so any assessment of garbage collection that used adaptive sizing prior to Java SE 8 will change because that implementation has changed.

One common tip says to set minimum heap to maximum heap size. You might want to do that sometimes. But often, today, you should not do this, because it inhibits the adaptive capabilities of the JVM. When the technology was less mature, it made sense because the JVM might not adapt well. But today it's much better at adapting, so we don't want to take that particular optimization off the table. If you do something that specifically runs counter to the optimizations that the JVM engineers have put into the JVM, you're doing yourself a disservice.
Java Magazine: What hardware changes should developers take into account?
Pepperdine: The biggest change has to be the multicore processors, which means we now have so much more compute capacity than we once had—which is not just based on clock speed but on our ability to scale out. The biggest software challenge is figuring out how to take advantage of this extra compute power. If you look at a computer itself, it's basically a box of things that aren't sharable. You can't share the bus, memory access, the CPU, the frame buffer, the video, and so on. This means that threads must compete for these resources, and with multicore processing, we can and do write applications that compete differently for all of these individual resources.

So you have to start building hardware differently or configuring it differently to make sure that you actually have enough capacity to support what your application is consuming. In addition, the change in the hardware is exposing bugs in our existing implementations, mostly because we made mistaken assumptions about how things are going to interact that show up as very subtle race conditions within the implementation.
Java Magazine: What are some impor-

Pepperdine checks in with clients from a café in Budapest.

tant misconceptions that you encounter about Java performance issues?
Pepperdine: People still don't understand JIT [just-in-time] technology very well. It's a systemic problem, because our educational systems come up with things like Omega notation, where people can look at an algorithm and make an assessment that it should run a certain way, which might work in a static environment. But as soon as you put that into the JVM in a JIT environment, all bets are off. The JIT can take the code and do so many different things to it that the underlying assembler code that gets executed will be functionally the same, but it might not look anything like what you wrote.

When people look at code, they'll do a static cost analysis of what it takes to execute the code but then they won't take into account what the JIT will do to that particular code, which means that their cost models based upon code evaluations are pretty much all wrong.
Java Magazine: You are quoted as saying that in identifying performance problems, the biggest mistake is to not consider the system as a whole. You claim that people miss important clues and sometimes fail to have proper tooling. Could you elaborate on these issues?
Pepperdine: I look at computing as a problem of resource management, which means you can apply a lot of economic principles to help you understand what's going on. And, the economics of the situation are governed by hardware, which is what makes things real. Hardware gives you real capacity and throughputs that are nonsharable. You need to know what resources your application is using, how many resources your application is consuming, and how the resources are being managed.

That gives you the biggest clues as to what your application is up to and what you can do to achieve a better balance in terms of resource utilization. When you get good balance, you'll get the best performance. Tooling tells you what your computer is up to. If you're lucky, the tooling will tell you which parts of your application are responsible for utilizing which parts of the available computing resources. This gives you a sense of what in your application needs to be changed in order to get better hardware utilization and, therefore, better performance for your end users.

It's also important to understand that hardware counters are strongly affected by what your application is doing, which is strongly affected by how your end users are actually using that code. The end user might be using the code in a way that's driving load down on the JVM, which is, of course, consuming hardware. So you have to consider the consumption patterns of the hardware. And if your users change how they're using your software, that's going to change how your application and how the JVM consume hardware. You have to look at this as an entire system to understand what's going on.

Java Magazine: What is the hardest problem you've encountered in performance tuning?

Pepperdine: I like hard problems—they're fun. The hardest problems have to do with low latency. I recently worked with a low-latency application where we had to meet a 60 hertz time budget. In other words, we had about 16.7 milliseconds in order to complete all the work. So the challenge was to understand what we were trying to do and how quickly we could get the work done. In the case of running Java, we have to understand when the JVM will kick in with a garbage collection cycle, how long the cycle is going to last, and whether it will cause us to miss the heartbeat that we have to deliver results on.

We wanted to look for different areas of instability, where we were getting outliers, and who was consuming what part of the 16.7 millisecond time budget. It's very challenging. We're testing different technologies to see which one will give us the most stable response time. Here, we're not interested in average response time. We're interested in what the primary response time is going to be and then how often and how much jitter is taking place ballooning the response time.

Java Magazine: How has the cloud changed performance tuning?

Pepperdine: It has changed the performance picture, because as long as you can scale out, your performance should remain stable. Clouds give you the ability to scale out in the virtualized hardware dimension, which you might not have in your current data center.

But it has also meant that monitoring has become a lot more important, because monitoring not only has to be able to tell you what's going on, with the cloud it has to be able to tell you when you need to expand or retract on capacity. In other words, if you want the elasticity to happen automatically, you need tooling that's going to be able to support that in terms of performance management.

So the tools have to become smarter in order to help people understand when they need to scale up and when they're able to back off. So the picture has changed. Before the cloud arrived, when something went off, you had to send a team in to find the problem, fix it, and bring the machine back up. But now, I'm not going to do that anymore. I'm going to spin up another virtual machine with the application running it and basically cycle the problematic node out and replace it with a new fresh one that I just put into the system. We want to give good, stable, consistent performance to end users, so that no matter how much chaos is going on behind the scenes, the users don't feel it.

ON THE CLOUD

There are many advantages to a cloud environment from an administrative and operational perspective. There's often a cost benefit in having a more elastic environment.

Java Magazine: What is your overall perspective of the cloud?

Pepperdine: The cloud is a way of virtualizing the hardware services to try to make them more containable and deployable. There are many advantages to a cloud environment from an administrative and operational perspective. And there's often a cost benefit in having a more elastic environment. In the old world, and the world most people are currently in, if you need a new machine, you ask procurement to buy something that you add to your cluster. Then you scale out your application to run across this new machine (or machines) that you've added to your cluster.

The list of Java performance tuning tips from five years ago that is no longer valid "is huge," says Pepperdine.

In a cloud environment, you can make your applications completely elastic, which means that as you need more capacity, you can just add it in, which comes down to having real physical infrastructure to support it. As long as I have that, I can scale up or scale down as much as I need to support my client base.

I've visited a number of clients who have taken this to a new extreme. They may be running on Amazon or one of the other popular providers, but they've taken their applications and made them about as elastic as is possible. Their applications can scale up or down in reaction to load, so if they need the extra capacity, they've got it—they're paying for it. If they don't need the extra capacity, they can pull back, and then they're not expending all of that money on unutilized capacity. The cloud is really a game-changer in the sense that we now have to architect systems that can fit into this new way of doing things.

That said, the cloud is not the proverbial silver bullet. If you don't have the hardware to support what you're doing, virtualized environments aren't going to help you.

Java Magazine: Pierre-Hugues Charbonneau, a Java EE consultant, has a list of the top causes of Java enterprise performance problems. Let me try out a few on you. First, "lack of proper capacity planning."

Pepperdine: Yes, I see that a lot. I don't know if it's overall capacity planning or just capacity planning in the sense of configuration of individual components within the system, but we certainly see a lot of that.

Java Magazine: How about "excessive Java VM garbage collections"?

Pepperdine: That's an issue near the top. There are a couple of issues: improper heap configuration—either internal sizes or max size—or perhaps the application itself is just not memory efficient. We've done work in both directions—and sometimes both at the same time.

Java Magazine: Next, "too many or too poor integration with external systems."

Pepperdine: That's an interesting problem. Most people basically just use a fire-and-wait type technique for that. And sometimes that's not really an adequate way to deal with things. You need something to surround the use of the external system, so that you can actually have good failover—planned failure failover and recovery.

Java Magazine: What about "lack of proper database SQL tuning and capacity planning"?

WHAT'S GOING ON?
You need to know what resources your application is using, how many resources your application is consuming, and how the resources are being managed.

The cloud is really a game-changer in the sense of how systems now have to be architected, says Pepperdine.

Pepperdine: Oh yes, that's a big problem. This is where investing in a good DBA is really going to help you. We always recommend that you get a good DBA, work with them, and really check out what's going on in that direction. A good portion of our consulting has been looking at interactions with a database.

Java Magazine: What about "Java EE middleware tuning problems"?

Pepperdine: Yes, with a number of clients, all they want us to do is tune the middleware to try to make it function better. It might be a matter of just getting thread pool sizes properly done, or doing some other tweaking. It really depends on the product you're using as to how you're going to tweak it. Generally, what we see with middleware problems is that virtualized environments tend to be shy in network capacity. How middleware functions is really highly dependent on networking capacity and latencies and things like that.

Java Magazine: How about "insufficient proactive monitoring"?

Pepperdine: That's a big one—a lot of people wait until things crash before they figure out something has gone wrong, and then they're just basically running around trying to fix it afterward.

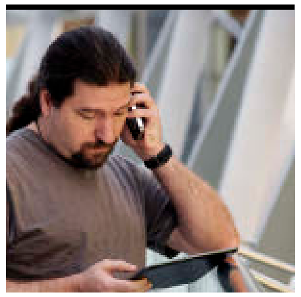
Java Magazine: "Saturated hardware on common infrastructure."

Pepperdine: Yes, we see this more and more with virtualization. From a hardware point of view, there's no such thing as threads; there's no such thing as processes; there's no such thing as any of these things that we think about. It's just streams of instructions and streams of data. So if you have one application misbehaving by consuming all the available bandwidth, that box is cooked for everything else running on it needing to use the network. You need to consider aggregate utilization of the available capacity. To that end, what we see people using more and more is network-attached storage. This is something that will completely unbalance a normally configured system. `</article>`

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

LEARN MORE

• Kodewerk Website



TED NEWARD
BIO

PHOTOGRAPH BY
PHIL SALTONSTALL

Part 1

Java 8: Lambdas

Get to know lambda expressions in Java 8.

Few things excite a community of software developers more than a new release of their chosen programming language or platform. Java developers are no exception. In fact, we're probably even more excited about new releases, partly because there was a time not too long ago when we thought that Java's fortunes—like those of Java's creator, Sun—were on the wane. A brush with death tends to make one cherish renewed life all the more. But in this case, our enthusiasm also stems from the fact that unlike the prior release, Java 8 will finally get a new "modern" language feature that many of us have been requesting for years—if not decades.

Of course, the major Java 8 buzz is around *lambdas* (also called *closures*), and that's where this two-part series will focus. But a language feature, on its own, will often appear anything but useful or interesting unless there's a certain amount of support behind it. Several features in Java 7 fit that description: enhanced numeric literals, for example, really couldn't get most people's eyes to light up.

In this case, however, not only do Java 8 function literals change a core part of the language, but they come alongside some additional language features designed to make them easier to use, as well as some library revamping that makes use of those features directly. These will make our lives as Java developers easier.

Java Magazine has run articles on lambdas before, but given that syntax and semantics might have changed since then and not all readers will have the time or inclination to read those articles, I will assume that readers have never seen any of this syntax before.

Note: This article is based on a prerelease version of Java SE 8 and, as such, it might not be entirely accurate when the final release ships. Syntax and semantics are always subject to change until the final release.

For those who desire a deeper, more official explanation of this material, Brian Goetz' papers, for example his "State of the Lambda: Libraries Edition" paper and others available at the Project Lambda home page, are priceless references.

CODE = OBJECT

As Java grew and matured, we found more places where treating blocks of code as objects (data, really) was not only useful but necessary.

Background: Functors

Java has always had a need for *functional objects* (sometimes called *functors*), though we in the community struggled mightily to downplay their usefulness and need. In Java's early days, when building GUIs, we needed blocks of code to respond to user events such as windows opening and closing, button presses, and scrollbar movement.

In Java 1.0, Abstract Window Toolkit (AWT) applications were expected, like their C++ predecessors, to extend window classes and override the event method of choice; this was deemed unwieldy and unworkable. So in Java 1.1, Sun gave us a set of "listener" interfaces, each with one or more methods corresponding to an event within the GUI.

But in order to make it easier to write the classes that must implement these interfaces and their corresponding methods, Sun gave us inner classes, including the ability to write such a class within the body of an existing class without having to specify a name—the ubiquitous *anonymous inner class*. (By the way, the listeners were hardly the only example of these that appeared during Java's history. As we'll see later, other, more "core" interfaces just like them appeared, for example, Runnable and Comparator.)

Inner classes had some strangeness to them, both in terms of syntax and semantics. For example, an inner class was either a *static inner class* or an *instance inner class*, depending not on any particular keyword (though static inner classes could be explicitly stated as such using the static keyword) but on the lexical context in which the instance was created. What that meant, in practical terms, is that Java developers often got questions such as those in **Listings 1a** and **1b** wrong on programming interviews.

"Features" such as inner classes often convinced Java developers that such functionality was best relegated to the corner cases of the language, suitable for a programming interview and not much else—except when they needed them. Even then, most of the time, they were used purely for event-handling reasons.

Above and Beyond

As clunky as the syntax and semantics were, however, the system worked. As Java grew and matured, we found more places where treating blocks of code as objects (data, really) was not only useful but necessary. The revamped security system in Java SE 1.2 found it useful to pass in a block of code to execute under a different security context. The revamped Collection classes that came with that same release found it useful to pass in a block of code in order to know how to impose a sort order on a sorted collection. Swing found it useful to pass in a block of code in order to decide which files to display to the user in a File Open or File Save dialog box. And so on. It worked—though often through syntax that only a mother could love.

But when concepts of functional programming began to enter mainstream programming, even Mom gave up. Though possible (see this remarkably complete example), functional programming in Java was, by any account, obtuse and awkward. Java needed to grow up and join the host of mainstream programming languages that offer first-class

LISTING 1a / LISTING 1b / LISTING 2

```
class InstanceOuter {
  public InstanceOuter(int xx) { x = xx; }

  private int x;

  class InstanceInner {
    public void printSomething() {
      System.out.println("The value of x in my outer is " + x);
    }
  }
}

class StaticOuter {
  private static int x = 24;
```

Download all listings in this issue as text

language support for defining, passing, and storing blocks of code for later execution.

Java 8: Lambdas, Target Typing, and Lexical Scoping

Java 8 introduces several new language features designed to make it easier to write such blocks of code—the key feature being *lambda expressions*, also colloquially referred to as *closures* (for reasons we'll discuss later) or *anonymous methods*. Let's take these one at a time.

Lambda expressions. Fundamentally, a lambda expression is just a shorter way of writing an implementation of a method for later execution. Thus, while we used to define a Runnable as shown in **Listing 2**, which uses the anonymous inner class syntax and clearly suffers from a "vertical problem" (meaning that the code takes too many lines to express the basic concept), the Java 8 lambda syntax allows us to write the code as shown in **Listing 3**.

Both approaches have the same effect: a Runnable-implementing object whose run() method is being invoked to print something to the console. Under the hood, however, the Java 8 version is doing a little more than just generating an anonymous class that implements the Runnable interface—some of which has to do with the invoke dynamic bytecode that was introduced in Java 7. We won't get to that level of detail here, but know that this is more than "just" an anonymous class instance.

Functional interfaces. The Runnable interface—like the Callable<T> interface, the Comparator<T> interface, and a whole host of other interfaces already defined within Java—is what Java 8 calls a *functional interface*: it is an interface that requires exactly one method to be implemented in order to satisfy the requirements of the interface. This is how the syntax achieves its brevity, because there is no ambiguity around which method of the interface the lambda is trying to define.

The designers of Java 8 have chosen to give us an annotation, @FunctionalInterface, to serve as a documentation hint that an interface is designed to be used with lambdas, but the compiler does not require this—it determines "functional interfaceness" from the structure of the interface, not from the annotation.

Throughout the rest of this article, we'll continue to use the Runnable and Comparator<T> interfaces as working examples, but there is nothing particularly special about them, except that they adhere to this functional interface single-method restriction. Any developer can, at any time, define a new functional interface—such as the following one—that will be the interface target type for a lambda.

```
interface Something {
  public String doit(Integer i);
}
```

The Something interface is every bit as legal and legitimate a functional interface as Runnable or Comparator<T>; we'll look at it again after getting some lambda syntax under our belt.

Syntax. A lambda in Java essentially consists of three parts: a parenthesized set of parameters, an arrow, and then a body, which can either be a single expression or a block of Java code. In the case of the example shown in **Listing 2**, run takes no parameters and returns void, so there are no parameters and no return value. A Comparator<T>-based example, however, highlights this syntax a little more obviously, as shown in **Listing 4**. Remember that Comparator takes two strings and returns an integer whose value is negative (for "less than"), positive (for "greater than"), and zero (for "equal").

If the body of the lambda requires more than one expression, the value returned from the expression can be handed back via the return keyword, just as with any block of Java code (see **Listing 5**). (Where we put the curly braces in code such as **Listing 5** will likely dominate Java message boards and blogs for years to come.) There are a few restrictions on what can be done in the body of the lambda, most of which are pretty intuitive—a lambda body can't "break" or "continue" out of the lambda, and if the lambda returns a value, every code path must return a value or throw an exception, and so on. These are much the same rules as for a standard Java method, so they shouldn't be too surprising.

Type inference. One of the features that some other languages have been touting is the idea of *type inference*: that the compiler should be smart enough to figure out what the type parameters should be, rather than forcing the developer to retype the parameters.

Such is the case with the Comparator example in **Listing 5**. If the target type is a Comparator<String>, the objects

LISTING 3 LISTING 4 LISTING 5

```
public static void main(String... args) {
  Runnable r2 = () -> System.out.println("Howdy, world!");
  r2.run();
}
```

Download all listings in this issue as text

passed in to the lambda *must* be strings (or some subtype); otherwise, the code wouldn't compile in the first place. (This isn't new, by the way—this is "Inheritance 101.")

In this case, then, the String declarations in front of the lhs and rhs parameters are entirely redundant and, thanks to Java 8's enhanced type inference features, they are entirely optional (see **Listing 6**).

The language specification will have precise rules as to when explicit lambda formal type declarations are needed, but for the most part, it's proving to be the default, rather than the exception, that the parameter type declarations for a lambda expression can be left out completely.

SCOPE IT OUT

Lambdas are lexically scoped, meaning that a lambda recognizes the immediate environment around its definition as the next outermost scope.

One interesting side effect of Java's lambda syntax is that for the first time in Java's history, we find something that cannot be assigned to a reference of type Object (see **Listing 7**)—at least not without some help.

The compiler will complain that Object is not a functional interface, though the real problem is that the compiler can't quite figure out which functional interface this lambda should implement: Runnable or something else? We can help the compiler with, as always, a cast, as shown in **Listing 8**.

Recall from earlier that lambda syntax works with any interface, so a lambda that is inferred to a custom interface will also be inferred just as easily, as shown in **Listing 9**. Primitive types are equally as viable as their wrapper types in a lambda type signature, by the way.

Again, none of this is really new; Java 8 is just applying Java's long-standing principles, patterns, and syntax to a new feature. Spending a few minutes exploring type inference in code will make that clearer, if it's not clear already.

Lexical scoping. One thing that is new, however, is how the compiler treats names (identifiers) within the body of a lambda compared to how it treated them in an inner class. Consider the inner class example shown in **Listing 10** for a moment.

When run, the code in **Listing 10** counterintuitively produces "Hello$1@f7ce53" on my machine. The reason for this is simple to understand: both the keyword this and the call to toString in the implementation of the anonymous Runnable are bound to the anonymous inner class implementation, because that is the innermost scope that satisfies the expression.

If we wanted (as the example seems to imply) to print out Hello's version of toString, we have to explicitly qualify it using the "outer this" syntax from the inner classes portion of the Java spec, as shown in **Listing 11**. How's that for intuitive?

Frankly, this is one area where inner classes simply created more confusion than they solved. Granted, as soon as the reason for the this keyword showing up in this rather unintuitive syntax was explained, it sort of made sense, but it made sense in the same way that politicians' perks make sense.

Lambdas, however, are *lexically scoped*, meaning that a lambda recognizes the immediate environment around its definition as the next outermost scope. So

LISTING 6 / LISTING 7 / LISTING 8 / LISTING 9 / LISTING 10 / LISTING 11

```
public static void main(String... args) {
  Comparator<String> c =
    (lhs, rhs) ->
      {
        System.out.println("I am comparing" +
                           lhs + " to " + rhs);
        return lhs.compareTo(rhs);
      };
  int result = c.compare("Hello", "World");
}
```

Download all listings in this issue as text

the lambda example in **Listing 12** produces the same results as the second Hello nested class example in **Listing 11**, but with a much more intuitive syntax.

This means, by the way, that this no longer refers to the lambda itself, which might be important in certain cases—but those cases are few and far between. What's more, if such a case does arise (for example, perhaps the lambda needs to return a lambda and it wants to return itself), there's a relatively easy workaround, which we'll get to in a second.

Variable capture. Part of the reason that lambdas are called *closures* is that a function literal (such as what we've been writing) can "close over" variable references that are outside the body of the function literal in the enclosing scope (which, in the case of Java, would typically be the method in which the lambda is being defined). Inner classes could do this, too, but of all the subjects that frustrated Java developers the most about inner classes, the fact that inner classes could reference only "final" variables from the enclosing scope was near the top.

Lambdas relax this restriction, but only by a little: as long as the variable reference is "effectively final," meaning that it's final in all but name, a lambda can reference it (see **Listing 13**). Because message is never modified within the scope of the main method enclosing the lambda being defined, it is effectively final and, therefore, eligible to be referenced from within the Runnable lambda stored in r.

While on the surface this might sound like it's not much of anything, remember that the lambda semantics rules don't change the nature of Java as a whole—objects on the other side of a reference are still accessible and modifiable long after the lambda's definition, as shown in **Listing 14**.

Astute developers familiar with the syntax and semantics of older inner classes will remember that this was also true of references declared "final" that were referenced within an inner class—the final modifier applied only to the reference, not to the object on the other side of the reference. Whether this is a bug or a feature in the eyes of the Java community remains to be seen, but it is what it is, and developers would be wise to understand how lambda variable capture works, lest a surprise bug appear. (In truth, this behavior isn't new—it's just recasting existing functionality of Java in fewer keystrokes and with more support from the compiler.)

Method references. Thus far, all the lambdas we've examined have been anonymous literals—essentially, defining the lambda right at the point of use. This is great for one-off kinds of behavior, but it doesn't really help much when that behavior is needed or wanted in several places. Consider, for example, the following Person class. (Ignore the lack of proper encapsulation for the moment.)

```
class Person {
  public String firstName;
  public String lastName;
  public int age;
});
```

When a Person is put into a SortedSet or needs to be sorted in a list of some form, we want to have different mechanisms by which Person instances can be sorted—for example, sometimes by first name and sometimes by last name. This is what Comparator<T> is for: to allow us to define an imposed ordering by passing in the Comparator<T> instance.

Lambdas certainly make it easier to write the sort code, as shown in **Listing 15**. But sorting Person instances by first name is something that might need to be done many times in the codebase, and writing that sort of algorithm multiple times is clearly a violation of the Don't Repeat Yourself (DRY) principle.

The Comparator can certainly be captured as a member of Person

LISTING 12 / LISTING 13 / LISTING 14 / LISTING 15

```
class Hello {
  public Runnable r = () -> {
    System.out.println(this);
    System.out.println(toString());
  };

  public String toString() {
    return "Hello's custom toString()";
  }
}
```

Download all listings in this issue as text

itself, as shown in **Listing 16**. The Comparator<T> could then just be referenced as any other static field could be referenced, as shown in **Listing 17**. And, truthfully, functional programming zealots will prefer this style, because it allows for the functionality to be combined in various ways.

But it feels strange to the traditional Java developer, as opposed to simply creating a method that fits the signature of Comparator<T> and then using that directly—which is exactly what a method reference allows (see **Listing 18**). Notice the double-colon method-naming style, which tells the compiler that the method compareFirstNames, defined on Person, should be used here, rather than a method literal.

Another way to do this, for those who are curious, would be to use the compareFirstNames method to create a Comparator<Person> instance, like this:

```
Comparator<Person> cf =
  Person::compareFirstNames;
```

And, just to be even more succinct, we could avoid some of the syntactic overhead entirely by making use of some of the new library features to write the following, which makes use of a higher-order function (meaning, roughly, a function that passes around functions) to essentially avoid all of the previous code in favor of a one-line in-place usage:

```
Arrays.sort(people, comparing(
  Person::getFirstName));
```

This, in part, is why lambdas, and the functional programming techniques that come with them, are so powerful.

Virtual extension methods. One of the drawbacks frequently cited about interfaces, however, is that they have no default implementation, even when that implementation is ridiculously obvious. Consider, for example, a fictitious Relational interface, which defines a series of methods to mimic relational methods (greater than, less than, greater than or equal to, and so on). As soon as any one of those methods is defined, it's easy to see how the others could all be defined in terms of the first one. In fact, all of them could be defined in terms of a Comparable<T>'s compare method, if the definition of the compare method were known ahead of time. But interfaces cannot have default behavior, and an abstract class is still a class and occupies any potential subclass' one implementation-inheritance slot.

With Java 8, however, as these function literals become more widespread, it becomes more important to be able to specify default behavior without losing the "interfaceness" of the interface. Thus, Java 8 now introduces *virtual extension methods* (which used to be known in a previous draft as *defender methods*), essentially allowing an interface to specify a default behavior for a method, if none is provided in a derived implementation.

Consider, for a moment, the Iterator interface. Currently, it has three methods (hasNext, next, and remove), and each must be defined. But an ability to "skip" the next object in the iteration stream might be helpful. And because the Iterator's implementation is easily defined in terms of the other three, we can provide it, as shown in **Listing 19**.

Some within the Java community will scream, claiming that this

LISTING 16 / LISTING 17 / LISTING 18 / LISTING 19

```
class Person {
  public String firstName;
  public String lastName;
  public int age;

  public final static Comparator<Person> compareFirstName =
    (lhs, rhs) -> lhs.firstName.compareTo(rhs.firstName);

  public final static Comparator<Person> compareLastName =
    (lhs, rhs) -> lhs.lastName.compareTo(rhs.lastName);

  public Person(String f, String l, int a) {
    firstName = f; lastName = l; age = a;
  }

  public String toString() {
    return "[Person: firstName:" + firstName + " " +
      "lastName:" + lastName + " " +
      "age:" + age + "]";
  }
}
```

Download all listings in this issue as text

is just a mechanism to weaken the declarative power of interfaces and create a scheme that allows for multiple inheritance in Java. To a degree, this is the case, particularly because the rules around precedence of default implementations (in the event that a class implements more than one interface with different default implementations of the same method) will require significant study.

But as the name implies, virtual extension methods provide a powerful mechanism for extending existing interfaces, without relegating the extensions to some kind of second-class status. Using this mechanism, Oracle can provide additional, powerful behavior for existing libraries without requiring developers to track different kinds of classes. There's no SkippingIterator class that developers now have to downcast to for those collections that support it. In fact, no code anywhere has to change, and all Iterator<T>s, no matter when they were written, will automatically have this skipping behavior.

It is through virtual extension methods that the vast majority of the changes that are happening in the Collection classes are coming. The good news is that your Collection classes are getting new behavior, and the even better news is that your code won't have to change an iota in the meantime. The bad news is that we have to defer that discussion to the next article in this series.

Conclusion

Lambdas will bring a lot of change to Java, both in terms of how Java code will be written and how it will be designed. Some of these changes, inspired by functional programming languages, will change the way Java programmers think about writing code—which is both an opportunity and a hassle.

We'll talk more about the impact these changes will have on the Java libraries in the next article in this series, and we'll spend a little bit of time talking about how these new APIs, interfaces, and classes open up some new design approaches that previously wouldn't have been practical due to the awkwardness of the inner classes syntax.

Java 8 is going to be a very interesting release. Strap in, it's going to be a rocket-ship ride. </article>

LEARN MORE

- Brian Goetz' "State of the Lambda: Libraries Edition" paper
- Project Lambda home page
- "Maurice Naftalin's Lambda FAQ"

JavaOne™

Sept. 22 - 26, San Francisco

REGISTER NOW

Save $200 by Sept. 20

Discover All Things Java

- Experience the latest advancements
- 500+ educational sessions
- Expand your development skills
- 100+ product demos
- Meet technology leaders

oracle.com/javaone

Gold Sponsor QUALCOMM

Bronze Sponsor ARM

ORACLE®

BEN EVANS

BIO

PHOTOGRAPH BY JOHN BLYTHE

Understanding the Java HotSpot VM Code Cache

Learn to detect and mitigate a full code cache.

Java HotSpot VM has an extremely advanced just-in-time (JIT) compiler, which enables Java HotSpot VM to produce very highly optimized machine code for any platform that Java HotSpot VM runs on.

In this article, we will lift the curtain on an important aspect of Java HotSpot VM's JIT compiler: the code cache. Understanding the code cache provides insight into a range of performance issues that are otherwise difficult to track down.

Note: Two previous articles in *Java Magazine*, "Introduction to JIT Compilation in Java HotSpot VM" and "Inside the Java HotSpot VM 2: Statistics for Performance Analysis," discuss introductory Java HotSpot VM and JIT compiler topics.

To start our journey toward the JIT compiler and code cache, let's start by considering the lifecycle of a Java method.

Lifecycle of a Java Method

The smallest unit of new code that the Java platform will load and link into a running program is a class. This means that when a new method is being onboarded, it must go through the class-loading process (as part of the class that contains it).

The class-loading process acts as a *pinch point*: a place where a lot of the Java platform's security checks are concentrated. The lifecycle of a Java method, therefore, starts with the class-loading process that brings a new class into the running Java Virtual Machine (JVM).

Class Loading

Class loading starts with a stream of bytes (often read from disk) that should be in the class file format. If the byte stream actually fits into the expected format, the class loader can attempt to link it.

The linking process has several phases, of which the first—and most important—is verification. This is the phase in which the JVM confirms that the new class file does not attempt to violate Java's robust programming model.

FULL HOUSE

What happens if the code cache fills? In short, compilation has to stop.

During the verification phase, a number of security constraints are checked. For example, it is verified that

- Methods respect access control keywords
- Methods are called with correct static types
- Variables are assigned only suitably typed values
- Variables are properly initialized before use

The bytecode of methods is also extensively checked.

A key point here is that the JVM is a stack machine.

This choice was a deliberate one—it is much easier to prove security (and other) properties on a stack machine as compared to a register-based machine.

This means that most of the checks we want to make on bytecode can be done economically via static analysis at class-loading time, which greatly reduces the chance of harmful code ever making it into a live JVM.

For example, the stack state can be deduced at every point in a method without needing to keep track of the contents of registers.

Note that for performance reasons, JDK classes (from rt.jar) are not checked. They are loaded by the primordial class loader, which doesn't do comprehensive security checks.

This use of class loading as the opportunity to verify bytecode slows down the class-loading process. However, the payoff is that it significantly speeds up runtime, because checks can be done once and then omitted when the code is actually run.

How Java HotSpot VM Implements Class Loading

The key method that is used to turn a stream of bytes into a class object is the Java method ClassLoader::defineClass(). This method delegates to a native method, ClassLoader::defineClass1(), which does some basic checks and string conversion and then calls a C function called JVM_DefineClassWithSource().

As we might expect, this is an entry point into the JVM, and it provides access into the C++ code of Java HotSpot VM. Java HotSpot VM uses the SystemDictionary to load a new class via the parseClassFile() method of ClassFileParser.

Once class loading has been completed, the bytecode of the method is placed inside a C++ object (methodOop), for the bytecode interpreter to use.

This is sometimes called the *method cache*, although the bytecode is actually held inline in the methodOop for performance reasons.

How Do Methods Get Compiled?

Java HotSpot VM maintains a large number of performance and tracing counters in the bytecode interpreter. These trigger the compilation of methods once the methods have been run 10,000 times (for the server compiler).

The code that is output from the compiler is machine code (specialized for the specific operating system and CPU in use). It is placed into a central place—the CodeCache (a C++ object)—which is a heap-like structure for holding CodeBlob instances (which are the compiled representations of method code).

With the code blobs in the code cache, the running system is then updated to use the new compiled code rather than interpreted mode (this is sometimes called *pointer swizzling*).

DID YOU KNOW?
In more-recent JDK versions (Java 7 Update 4 and later), there is an additional form of code cache flushing: **speculative flushing.**

PrintCompilation

One of the simplest flags that can be used to control the JIT compilation subsystem is -XX:+PrintCompilation. This switch tells the JIT threads to add compilation messages to the standard log. PrintCompilation was introduced in "Introduction to JIT Compilation in Java HotSpot VM."

Deoptimization

Java HotSpot VM's server mode uses optimizations that it can't always prove hold true. It protects these optimizations with sanity checks (often called *guard conditions*), and if a check fails, Java HotSpot VM will deoptimize the code that was based on that assumption.

It's very common for Java HotSpot VM to then reconsider and try an alternative optimization. This means that the same method might be deoptimized and recompiled several times.

We can see deoptimization events in the PrintCompilation log; they show up as lines such as "made not entrant" and "made zombie."

These lines mean that a particular method, which had been compiled to a code blob, has now been deoptimized. This usually (but not always) happens because a new class was loaded and invalidated an assumption made by Java HotSpot VM.

What Happens as the Program Warms Up?

After a Java program starts up and goes through its initialization phases, it will normally get into normal operation and the hot paths of code will start to develop.

If we do multiple runs with the PrintCompilation switch on and collect the logs of which methods were compiled, a pattern emerges:

- Compilation usually eventually stops.
- The number of compiled methods stabilizes.
- The set of compiled methods on the same platform for the same test inputs is usually fairly consistent.
- The exact details of which methods get compiled depend on the exact JVM, operating system platform, and CPU in use.

Note: The compiled code for a given method is not guaranteed to be even roughly the same size across platforms.

This is the usual pattern, but there are cases in which the picture can be different from this; you should always check. One good way to do this is with Java VisualVM, which shows the general shape of the class-loading curve in its Classes section (see **Figure 1**).

What Happens if the Code Cache Fills?

In short, compilation has to stop. This is because once a code blob is compiled, usually only deoptimization can remove it from the code cache.

Code cache space is reclaimed by flushing the "zombie" code blobs from the code cache. (Over time, any "not entrant" blobs turn into zombies as code returns from them.)

In more-recent JDK versions (Java 7 Update 4 and later), there is an additional form of code cache flushing: *speculative flushing*. In this approach, the older methods are marked as being potentially eligible for flushing and disconnected from the methodOop that created them. If the VM needs to call the compiled method, the method is relinked back to its methodOop and survives being flushed.

However, if the method is not called again sufficiently quickly, the methodOop is reverted to interpreted mode, and the code blob is eligible for being flushed.

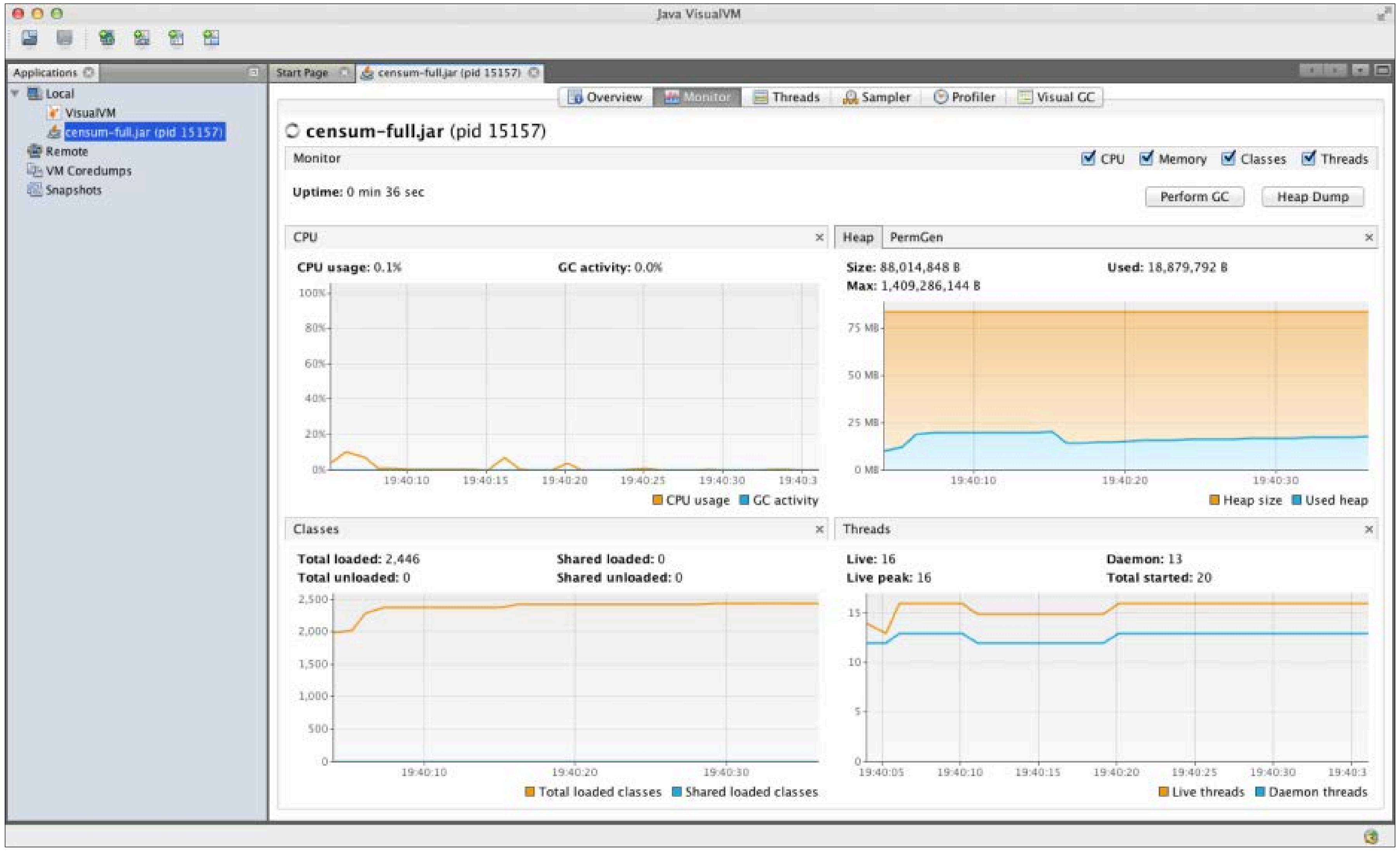


Figure 1

What Happens During Startup?

To see why application startup time could be problematic for the code cache, let's consider an imaginary Spring application.

Spring applications start up using the Bootstrap class, which locates an XML file detailing the instances to be created and wired up (and, hence, defines the classes to be loaded).

This means that Spring applications go through two phases of class loading: first, the phase of loading the classes needed to start the bootstrapping, and then a second phase that occurs when the application classes are typically loaded.

From the point of view of JIT compilation, this is important because the Spring framework uses reflection and other techniques to discover which classes to load and instantiate. These framework methods are called heavily during application startup but then are never touched again after that.

If a Spring framework method is run enough to be compiled, it is going to be of minimal use to the application. It will marginally improve application startup time, but at the price of using up a scarce resource. If enough framework methods are compiled, they can use up the entire code cache, leaving no room for the application methods that we actually want to be compiled.

To solve this problem, the JVM uses a system of counter decays. In their simplest form, these reduce the invocation counts for methods by 50 percent every 30 seconds.

This means that if methods are used only at startup, their invocation counts will, within a few minutes, have decayed down to effectively zero again. This prevents the rarely used Spring framework methods from using valuable code cache space. More generally, this technique protects against so-called *lukewarm methods*.

Which Switches Control Compilation and the Code Cache?

The following switches control compilation and the code cache:

- -XX:+PrintCompilation shows log entries for compilation and deoptimization events.
- -XX:CompileThreshold=n changes the number of times a method must be called before being compiled.
- -XX:ReservedCodeCacheSize=YYm sets the overall size of the code cache to be used.
- -XX:+UseCodeCacheFlushing allows a JVM to flush little-used code blobs (this is on by default in Java 7 Update 4 and above).

How Do We Detect Applications That Might Be Suffering from a Full Code Cache?

First prove that the cache is actually filling prematurely. Either show that the "compilation halted" message has been generated, or use the following process:

1. Use -XX:+PrintCompilation to output the methods that are actually being compiled.
2. Wait until this reaches steady state.
3. Repeat a few runs, and check that the results set is stable.
4. Try increasing the size of code cache (doubling is often a good first step) using -XX:ReservedCodeCacheSize. If more methods are now seen to be compiled, you can be sure that the original code cache was too small.
5. Retest overall performance to ensure that increasing the code cache size hasn't harmed some other aspect of application performance.

Conclusion

In this article, we explored the code cache, which stores JIT-compiled code in Java HotSpot VM. We saw how a method is loaded via class loading, and we saw the lifecycle of compilation, optimization, possible deoptimization, and flushing that methods undergo. We also discussed the problems that application startup can cause for compilation strategies and how Java HotSpot VM mitigates them.

We also explored some of the switches that are used to control compilation, and we discussed how to detect and mitigate a full code cache to ensure that your application code gains full benefit from JIT compilation. </article>

LEARN MORE

- "Introduction to JIT Compilation in Java HotSpot VM"
- "Inside the Java HotSpot VM 2: Statistics for Performance Analysis"

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 your members.

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

LEARN MORE

ORACLE®

NetBeans IDE 7.3.1: Out-of-the-Box Support for Java EE 7

Learn how to use NetBeans IDE 7.3.1 to take advantage of key Java EE 7 specifications.

GEERTJAN WIELENGA

BIO

*Watch author **Geertjan Wielenga** discuss using NetBeans IDE with Java EE 7.*

With Java EE 7, developers have more choice when developing their user interfaces, while also being able to leverage the power of HTML5. Java EE 7 helps developers code faster, with less boilerplate code, and it further standardizes the specifications making up the Java EE platform.

In this article, we'll take a look at the key Java EE 7 specifications and how NetBeans IDE 7.3.1 provides tools to help get you started using them.

Getting Started

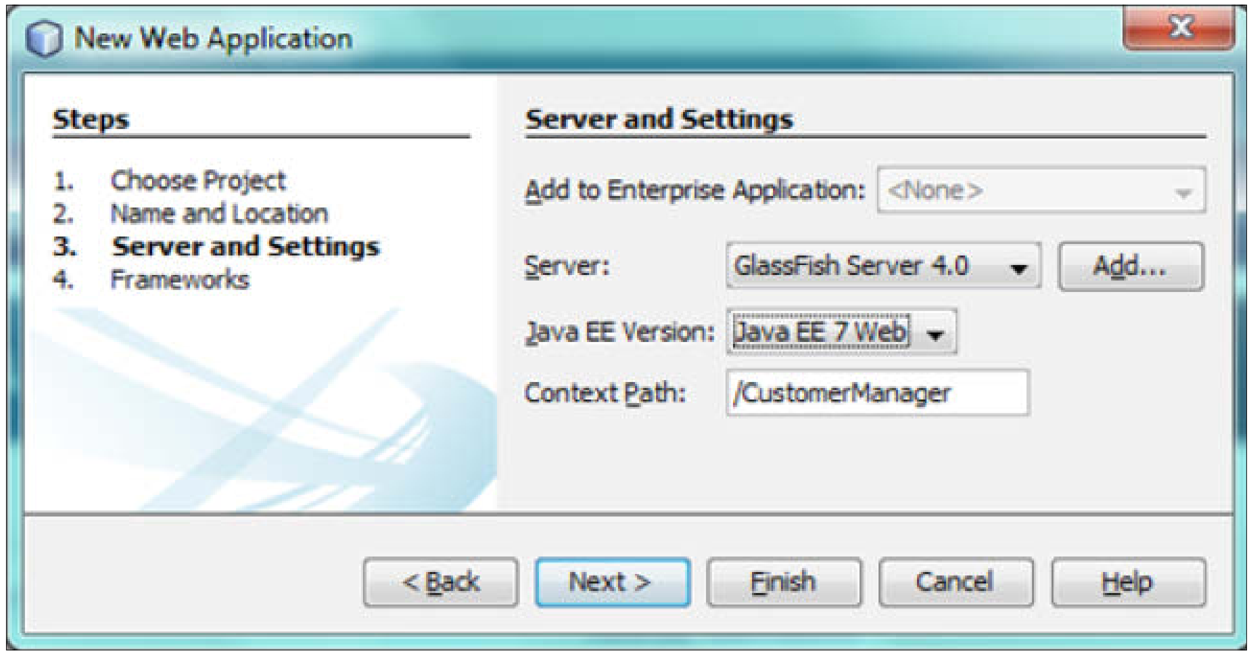
In the IDE, you can create applications conforming to the specifications that make up the Java EE 7 platform and specify that they be deployed to GlassFish 4, which is bundled with the IDE. Both the Ant-based and Maven-based Web project templates in NetBeans IDE enable you to get started with Java EE 7 and GlassFish 4 (see **Figure 1**).

Previously, you needed to specify that CDI, the Contexts and Dependency Injection framework, should be enabled in order to use its features. This is no longer necessary. When Java EE 7 is set as the application's Java EE platform version, CDI is automatically supported, and the required beans.xml configuration file is generated, as shown in **Figure 2**.

Moreover, the bundled Java EE Javadoc has been updated to Java EE 7, while new deployment descriptor formats and configuration file formats are recognized and supported by all Java EE features.

Sample Projects

To help get you started with Java EE 7, NetBeans IDE introduces a new set of samples specifically geared toward the Java EE 7 platform (see **Figure 3**). For example, do you want to know how WebSocket works in a real context? And what about JSON or the latest tips and tricks relating to the JavaServer Faces (JSF) expression language? There's nothing simpler than get-



New Web Application

Steps
1. Choose Project
2. Name and Location
3. **Server and Settings**
4. Frameworks

Server and Settings

Add to Enterprise Application: <None>
Server: GlassFish Server 4.0
Java EE Version: Java EE 7 Web
Context Path: /CustomerManager

Figure 1

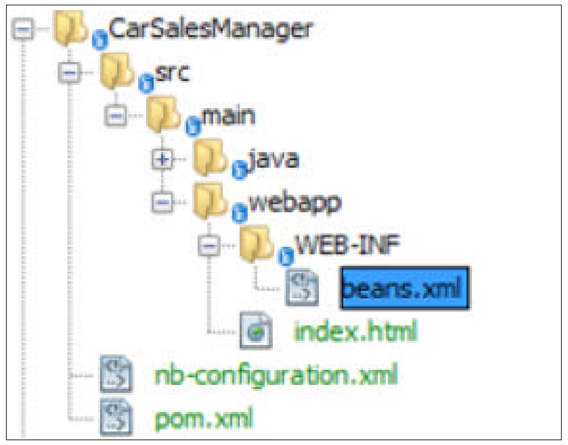
ting one of the many new sample applications and hacking the code that the IDE provides.

Templates and Wizards

Templates and wizards in the IDE have been rewritten to create code that conforms to the new Java EE 7 specifications. Code is now easier to read and maintain, and the previously required boilerplate has been removed.

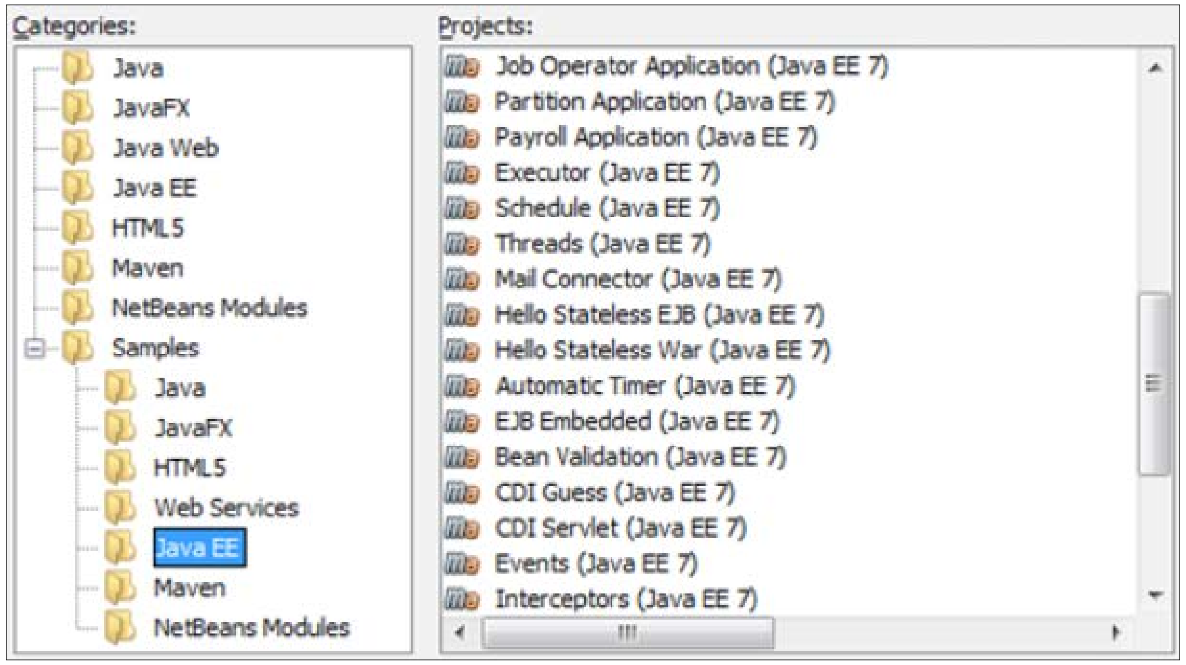
- **JAX-RS.** When you generate a Java-based JAX-RS client, the IDE now generates code that uses the new JAX-RS client API, version 2.0, which results in reduced, cleaner code. New wizards have been added for the newly defined JAX-RS Filter (see **Figure 4**) and Interceptor.

  The Cross-Origin Resource Sharing Filter wizard now generates standard JAX-RS code using the new JAX-RS Filter APIs, instead of the Jersey-specific code that was created previously. Finally, a new action has been added to the Insert Code dialog box to convert existing REST methods to asynchronous, via select **Insert Code** and then select



```
CarSalesManager
  src
    main
      java
      webapp
        WEB-INF
          beans.xml
        index.html
  nb-configuration.xml
  pom.xml
```

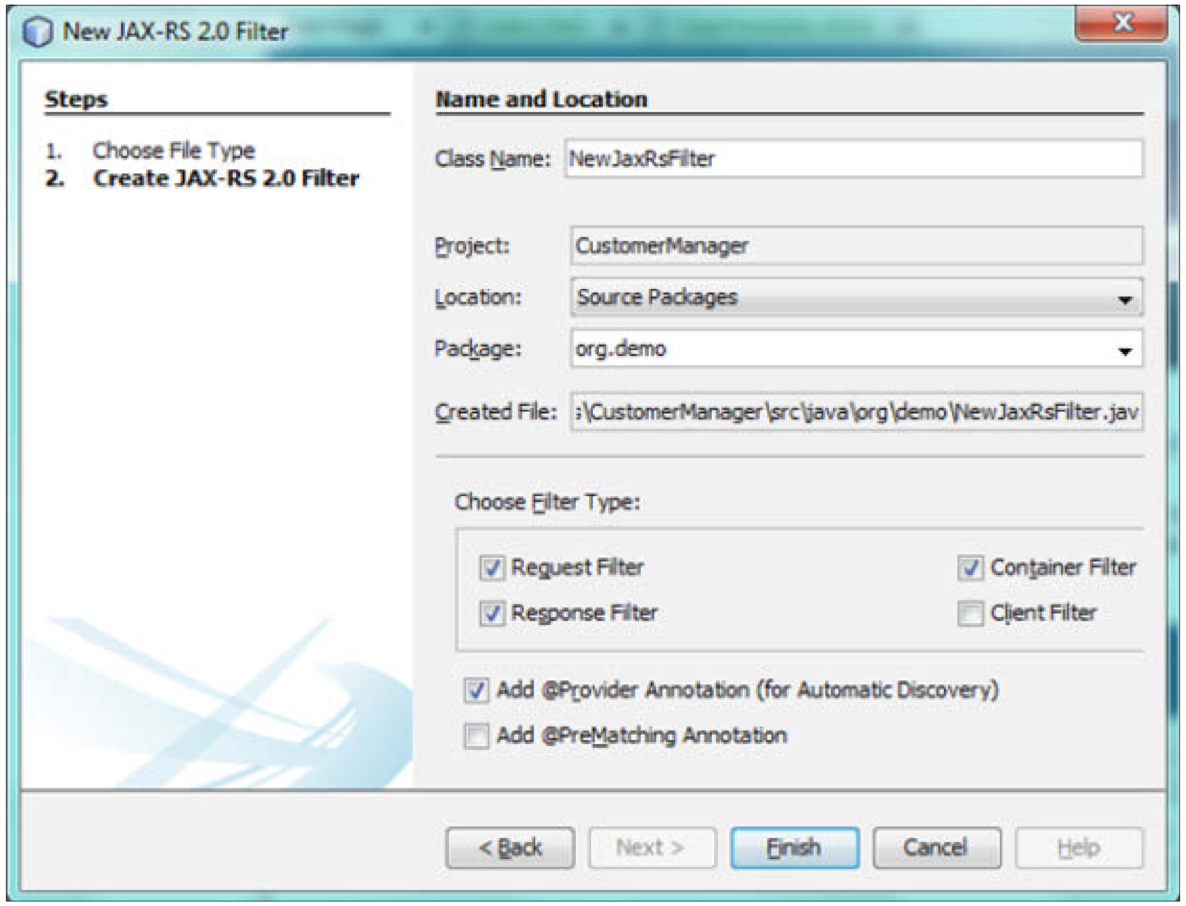
Figure 2



Categories:
- Java
- JavaFX
- Java Web
- Java EE
- HTML5
- Maven
- NetBeans Modules
- Samples
  - Java
  - JavaFX
  - HTML5
  - Web Services
  - Java EE
  - Maven
  - NetBeans Modules

Projects:
- Job Operator Application (Java EE 7)
- Partition Application (Java EE 7)
- Payroll Application (Java EE 7)
- Executor (Java EE 7)
- Schedule (Java EE 7)
- Threads (Java EE 7)
- Mail Connector (Java EE 7)
- Hello Stateless EJB (Java EE 7)
- Hello Stateless War (Java EE 7)
- Automatic Timer (Java EE 7)
- EJB Embedded (Java EE 7)
- Bean Validation (Java EE 7)
- CDI Guess (Java EE 7)
- CDI Servlet (Java EE 7)
- Events (Java EE 7)
- Interceptors (Java EE 7)

Figure 3



New JAX-RS 2.0 Filter

Steps
1. Choose File Type
2. **Create JAX-RS 2.0 Filter**

Name and Location

Class Name: NewJaxRsFilter

Project: CustomerManager

Location: Source Packages

Package: org.demo

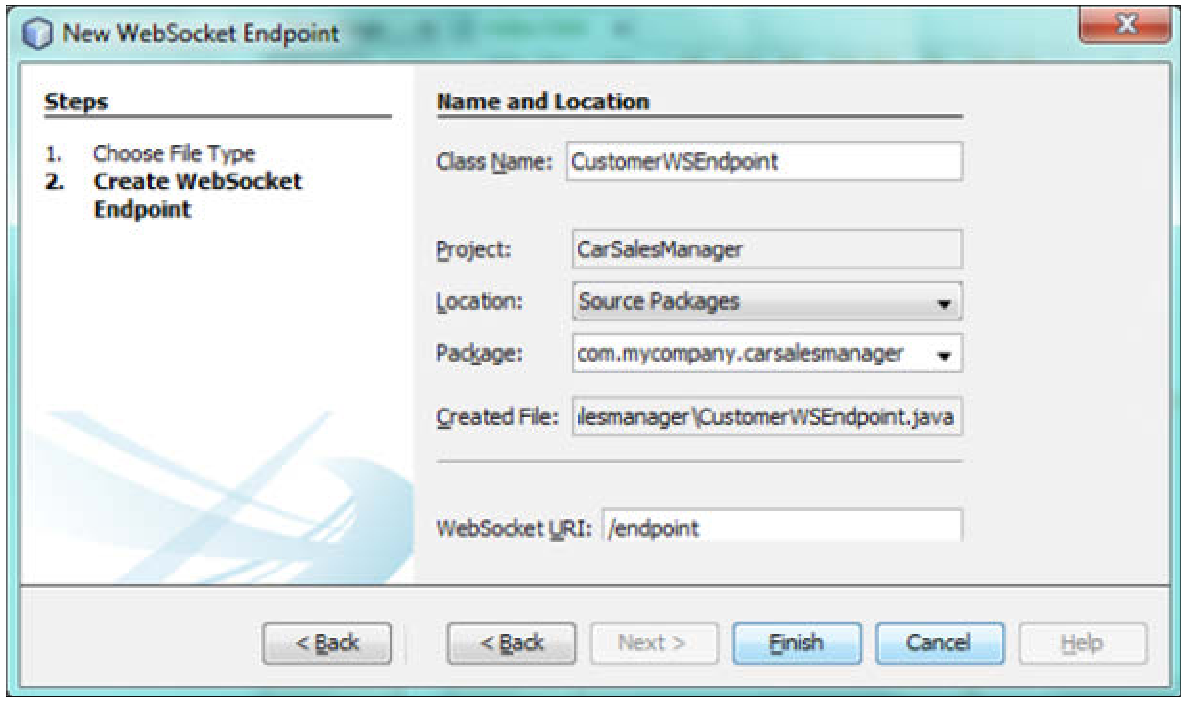
Created File: :\CustomerManager\src\java\org\demo\NewJaxRsFilter.jav

Choose Filter Type:
- [✓] Request Filter
- [✓] Container Filter
- [✓] Response Filter
- [ ] Client Filter

[✓] Add @Provider Annotation (for Automatic Discovery)

[ ] Add @PreMatching Annotation

Figure 4



New WebSocket Endpoint

Steps
1. Choose File Type
2. **Create WebSocket Endpoint**

Name and Location

Class Name: CustomerWSEndpoint

Project: CarSalesManager

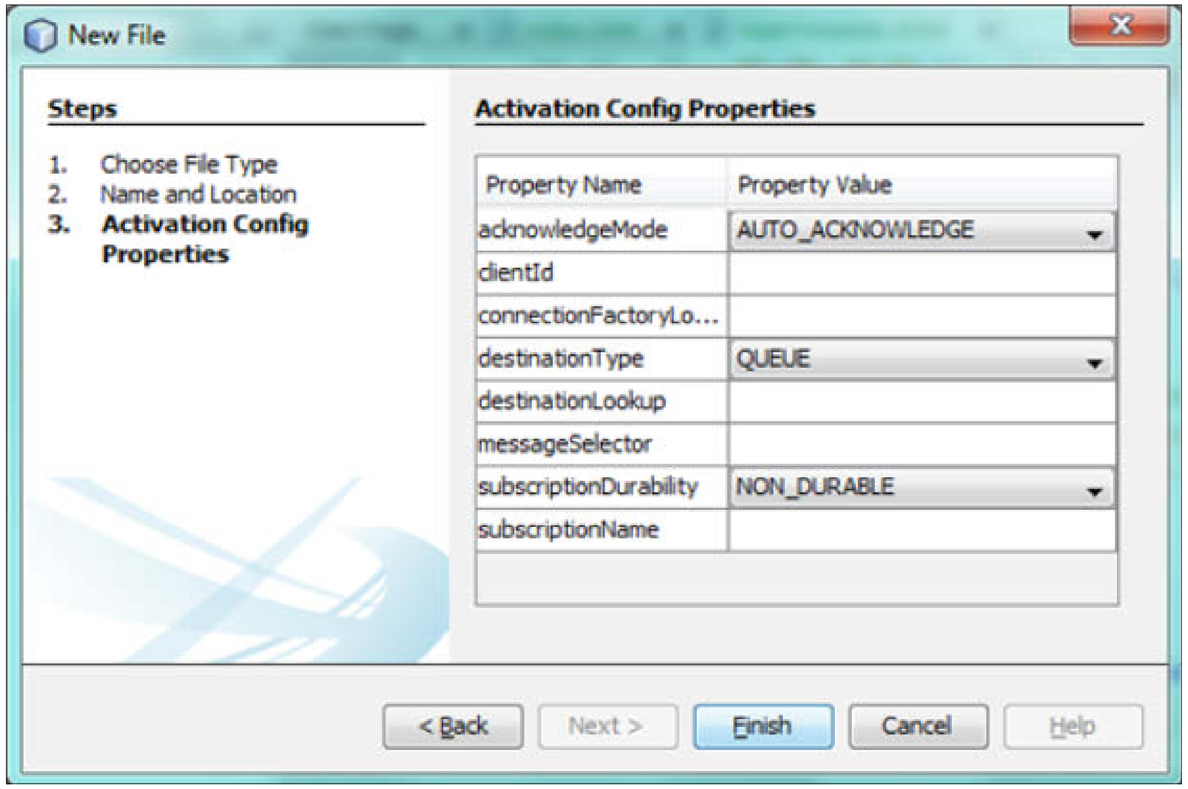
Location: Source Packages

Package: com.mycompany.carsalesmanager

Created File: lesmanager\CustomerWSEndpoint.java

WebSocket URI: /endpoint

Figure 5

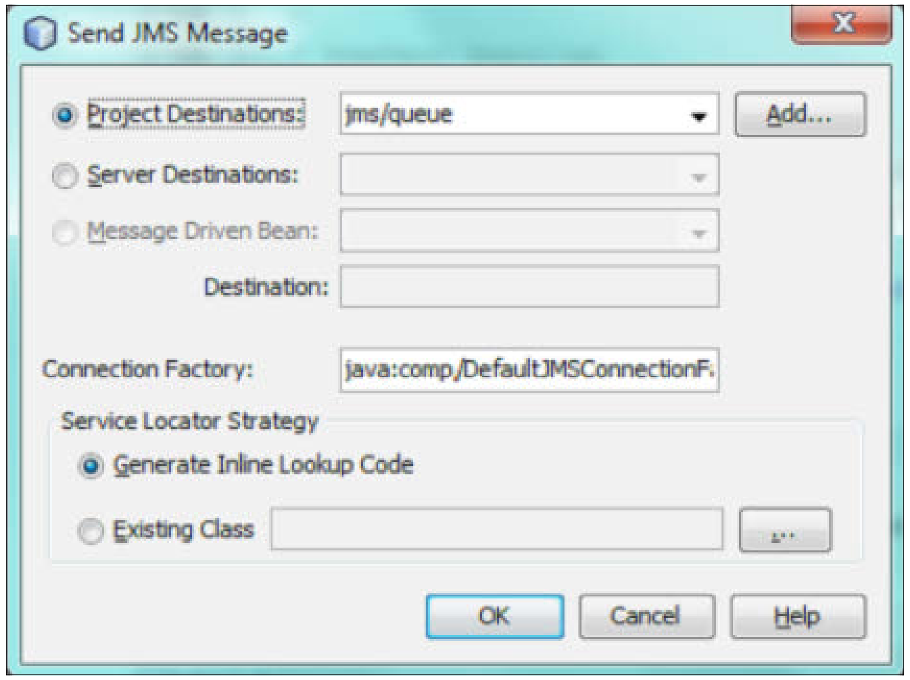

New File

Steps
1. Choose File Type
2. Name and Location
3. Activation Config Properties

Activation Config Properties

Property Name	Property Value
acknowledgeMode	AUTO_ACKNOWLEDGE
clientId	
connectionFactoryLo...	
destinationType	QUEUE
destinationLookup	
messageSelector	
subscriptionDurability	NON_DURABLE
subscriptionName	

Figure 6


Send JMS Message

Project Destinations: jms/queue
Server Destinations:
Message Driven Bean:
Destination:
Connection Factory: java:comp/DefaultJMSConnectionF
Service Locator Strategy
Generate Inline Lookup Code
Existing Class

Figure 7

Convert Method to Asynchronous.

- **WebSocket.** A new wizard has been added to create a WebSocket endpoint, as shown in **Figure 5**.
- **JPA.** Java Persistence API (JPA) version 2.1 is now supported, table names and ASC/DESC completion in @Index annotation have been added, and the bundled EclipseLink has been updated to version 2.5 with JPA 2.1 support.
- **JMS.** Java Message Service (JMS) version 2.0 is now supported. The Message-Driven Bean wizard now has a second page—Activation Config Properties—for specifying standard properties that are now defined by the specification (see **Figure 6**).
  The Send JMS Message action generates simplified, Java EE 7–specific code and enables CDI if necessary (see **Figure 7**).
- **EJB.** Enterprise JavaBeans (EJB) version 3.2 is now supported. Java EE 7 Web projects are able to use nonpersistent TimerSessionBeans, while new hints have been added to the IDE, such as "Incorrect usage of the @Asynchronous method invocation."
- **Expression Language.** The NetBeans IDE editor supports the new Expression Language (EL) 3.0 syntax and structures. Completion for EL 3.0 operators has been added on the Iterable properties and elements. There is now also completion for static methods, fields invocation, and additional support for resource bundles.
- **JSF.** JSF 2.2 is now supported and bundled with the IDE. The JSF Template Client wizard supports ResourceLibrary contracts in external JAR files or in the "contracts" folder of the WEB-ROOT. Support has been added for the new @FlowScoped beans in the editor and wizards. Tags and namespace definitions are now used in the @FacesComponent, while the bundled PrimeFaces framework has been upgraded to version 3.5. New hints have been added, such as hints indicating that an @FlowScoped bean is used in a project that is not capable of CDI.

Conclusion

NetBeans IDE is the ideal tool to use when you're learning about Java EE 7. It also enhances developer productivity for experienced Java EE 7 developers via its many templates and a variety of enhancements in its editor. </article>

LEARN MORE

- NetBeans IDE

DIERK KÖNIG

BIO

Enterprise JavaFX with OpenDolphin

Dolphin bridges the worlds of enterprise Java and desktop Java.

JavaFX technology creates stunning user interfaces. But how should a JavaFX enthusiast develop the next enterprise application or migrate an existing one? OpenDolphin is a free open source library that strictly separates business logic from visualization. This architectural approach assists in making a switch between different UI toolkits while ensuring that all investments in business logic are protected.

At JavaOne 2012, OpenDolphin was presented at the Java Strategy Keynote as an example of successfully integrating JavaFX with enterprise applications. The keynote demonstrated an interactive 3-D application that monitors a container yard, facilitates planning activities for container relocations, and simulates the execution of resulting work plans (see **Demo 1**).

This container monitoring demo—from Navis (CargoTec), the world market leader of container terminal software—illustrates two important characteristics of many JavaFX enterprise applications:

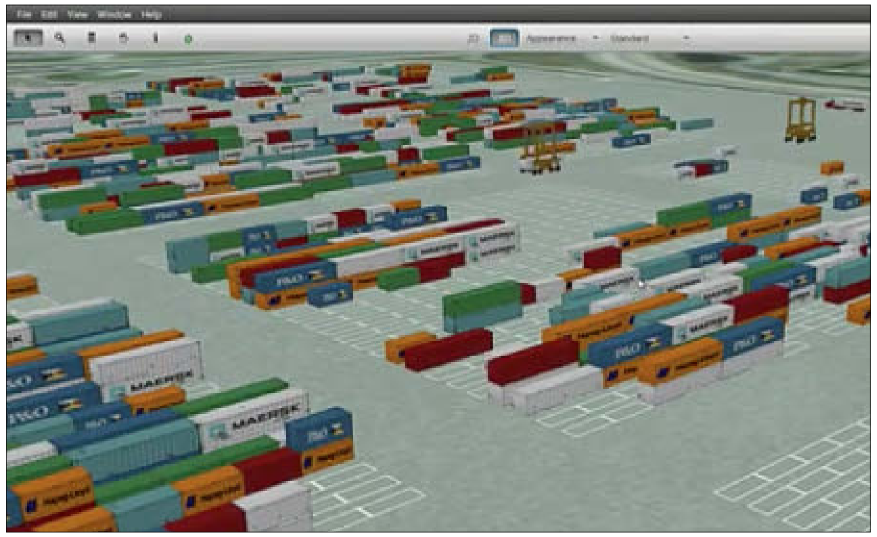
- First, you want to fully exploit the rich UI capabilities of JavaFX.
- Second, often a server-centric programming model has been used for many years and this architecture cannot be compromised.

Total Immersion

Not every business application needs the latest and greatest 3-D capability for exploring its data, even though that can be unexpectedly beneficial in many use cases. But an exciting application must at least accommodate the type of user experience we have become used to from smartphones and tablets, including context-sensitive input assistance, direct data manipulation, smooth transitions, and instant visualization of resulting effects. Anything less would not look or feel engaging.

Demo 2 shows a demo of a more conventional application that manages a financial portfolio. It is one of the many demos that ship with OpenDolphin. When you play the video, you will discover that even a business application that looks rather traditional at first can be charming if it offers the following:

- Instant, consistent update of all controls to visualize resulting values
- Smooth appearance and disappearance (no "slap in the face") of UI elements such as forms and dialog boxes
- Animated transitions and controls that "materialize" when they are needed



Demo 1: 3-D container yard demo that uses JavaFX

- Server round-trips that seem to take no time

The engaging user experience in the portfolio demo is achieved by using JavaFX and OpenDolphin together. OpenDolphin contributes the presentation model and, thus, determines *what* to display with instant, asynchronous updates from the server. The application programmer binds the JavaFX views of the application against these presentation models. The JavaFX views are responsible for *how* to display state and transitions.

The consistency of the displayed information and the snappy updates are features of OpenDolphin. Furthermore, the visualization code becomes more coherent and less coupled: no view needs to know about any other view. Neither is there a "presenter" object that depends on all views in the system. Each single view binds against a presentation model. That's it.

Mitigating Migration Risks

OpenDolphin is very flexible when it comes to supporting different UI toolkits. JavaFX is preferred, but you can also use Swing, AWT, SWT, and client-side frameworks such as Eclipse RCP or NetBeans. Any Java-based UI toolkit is just fine as long as it is able to register itself as a listener in a JavaBeans-like fashion.

Here's how to perform a UI toolkit migration: First encapsulate the view layer with the help of OpenDolphin, and then replace it. This strategy safeguards you regardless of whether you want to prepare yourself for a future move to JavaFX or you want to jump into JavaFX right away and hedge your bets.

HOW TO MIGRATE

To perform a UI toolkit migration, first encapsulate the view layer with the help of OpenDolphin and then replace it.

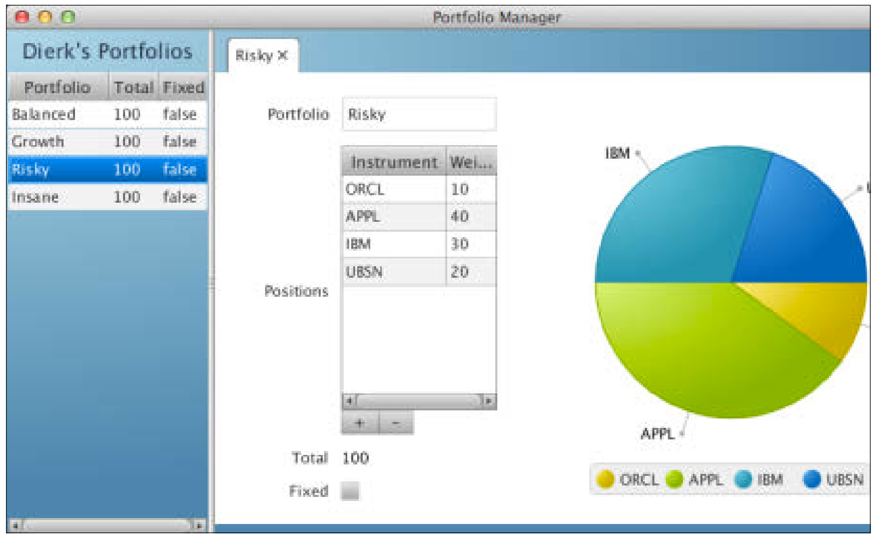
The demo section of OpenDolphin contains the so-called performance demo (see **Demo 3**), which comes in a Swing version and a JavaFX version to show the ease of migration. The business logic is not touched at all. The same application code runs on the server for both applications, and only the views differ.

Be Prepared for Change

Separating views makes it much easier to create, test, debug, and maintain them and—last but not least—adapt and extend them to meet changing requirements. Technological advances, version upgrades, and new toolkits might also trigger the need for change.

Sometimes, you just want to exploit new options. Take the container yard demo in **Demo 1**. It was first developed in full 2-D. When the JavaFX 3-D capability became available, the application team wanted to make use of that. The change was limited to the view layer, and we could even support switching between 2-D and 3-D mode at runtime.

When migrating, you not only have to care about the visual appearance but also about how the toolkit behaves in terms of concurrency. This is where



Portfolio	Total	Fixed
Balanced	100	false
Growth	100	false
Risky	100	false
Insane	100	false

Portfolio: Risky

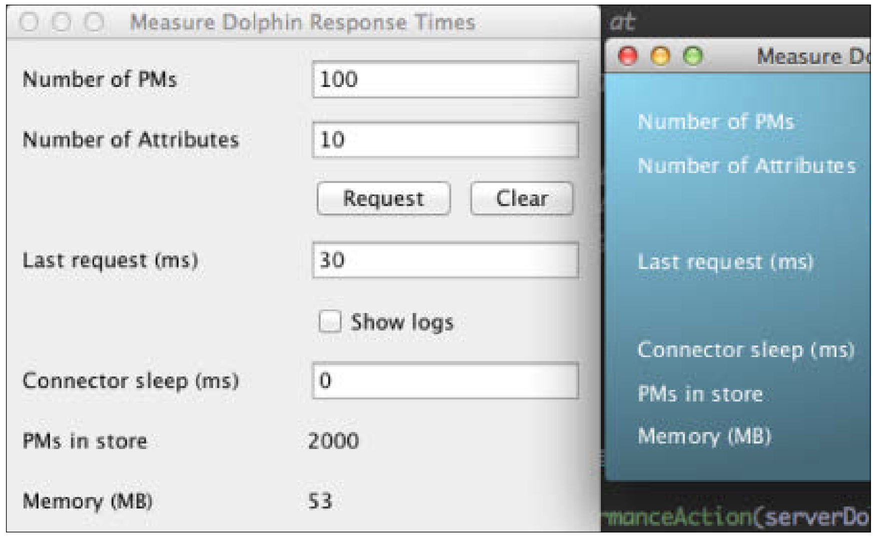
Positions:

Instrument	Wei...
ORCL	10
APPL	40
IBM	30
UBSN	20

Total: 100
Fixed: ☐

● ORCL ● APPL ● IBM ● UBSN

Demo 2: Portfolio management demo



Number of PMs: 100
Number of Attributes: 10
Last request (ms): 30
☐ Show logs
Connector sleep (ms): 0
PMs in store: 2000
Memory (MB): 53

```
manceAction(serverDo
```

Demo 3: Performance demo showing Swing-to-JavaFX migration

OpenDolphin shines again. Any access to a visual component is guaranteed to happen in the UI thread. Any controller action is guaranteed to be executed outside the UI thread. The UI thread is never blocked.

It is practically impossible to get the threading wrong because OpenDolphin enforces a strict distinction between view and controller objects in which no references are shared. They are not even on the same classpath.

Logical and Physical Separation

We've mainly discussed what happens on a single desktop machine, but enterprise applications run on the server. Therefore, OpenDolphin allows you to install all controlling logic on a physical server with automatic synchronization of client- and server-side presentation models. A sketch of this distributed architecture is shown in **Figure 1**.

The model-view-controller (MVC) separation in OpenDolphin is a logical separation—with the *M* denoting not a domain model but a presentation model. View and controller are always strictly separated, they run in different threads, and they even compile independently. They are typically even distributed on different machines: the view on the client and the controller on the server. For the purposes of developing, testing, and debugging, they can optionally reside inside the same Java Virtual Machine (JVM).

With the view running on the client, the view can fully leverage the fidelity of JavaFX. The server-side controller puts the business logic in the enterprise context, where it is centrally stored and managed and can access corporate datasources as fast as possible.

The server also serves as the hub when synchronizing multiple clients. The train control demo (see **Demo 4**) and the many events demo (see **Demo 5**) use this feature heavily.

These demos again give the impression that server round-trips take no time. The trick is to call the server asynchronously and to send only small bits of data. A typical round-trip is faster than 50 milliseconds and because of the asynchronicity, there is no user waiting time.

The only information that the client and server exchange is a limited set of commands. The command pattern allows us to batch up commands when the

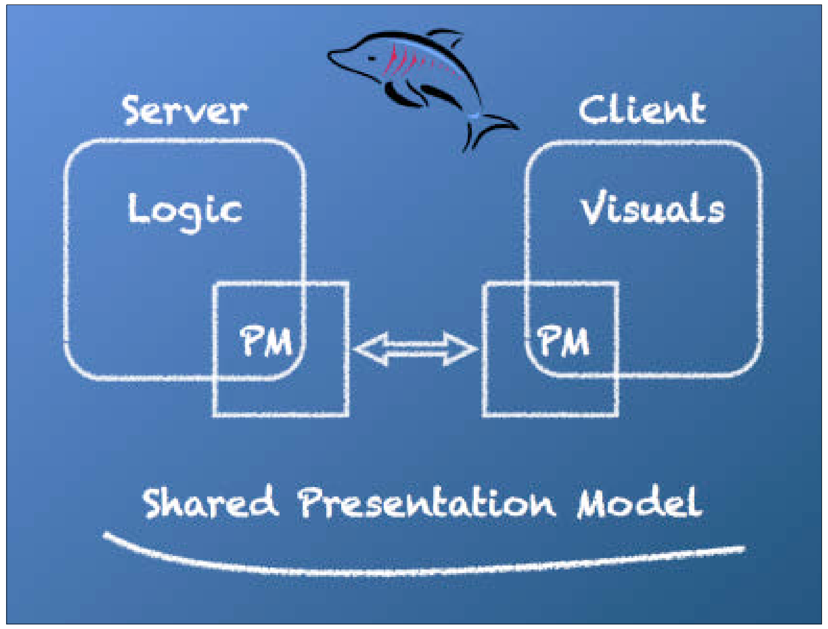
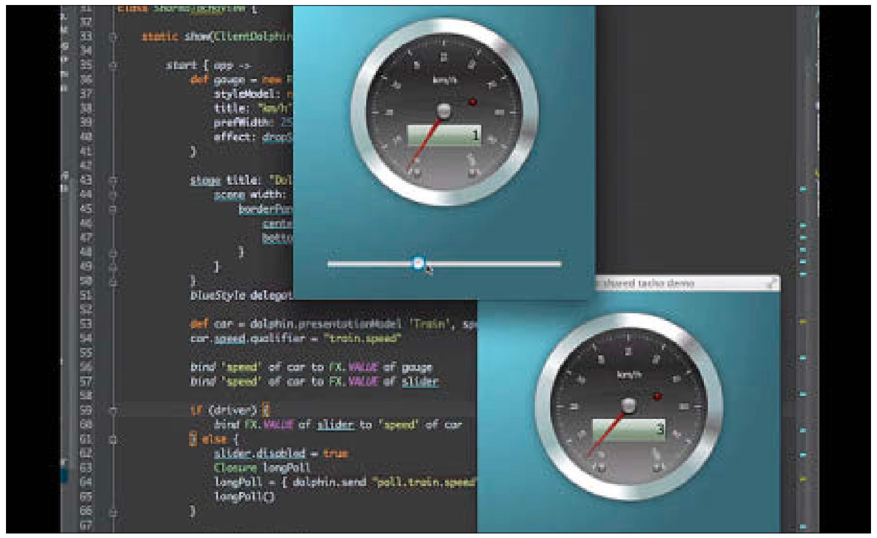


Figure 1



```
class ...
    static show(ClientDolphin...
        start { app ->
            def gauge = new ...
                styleModel: ...
                title: "km/h"...
                prefWidth: ...
                effect: dropS...
            }
            stage title: "Dol...
                scene width: ...
                    borderPan...
                        cent...
                        botto...
                    }
                }
            }
            BlueStyle delegat...

            def car = dolphin.presentationModel 'Train', sp...
            car.speed.qualifier = "train.speed"

            bind 'speed' of car to FX.VALUE of gauge
            bind 'speed' of car to FX.VALUE of slider

            if (driver) {
                bind FX.VALUE of slider to 'speed' of car
            } else {
                slider.disabled = true
                Closure longPoll
                longPoll = { dolphin.send "poll.train.speed"...
                longPoll()
            }
```

Demo 4: Train control demo

Demo 5: Many events demo

server is temporarily unavailable, apply optimizations, replay, and even be ready to support undo and redo operations.

In addition, the actual transport mechanism, protocol, and format are pluggable. The transport mechanisms that are currently supported are HTTP REST and in-memory.

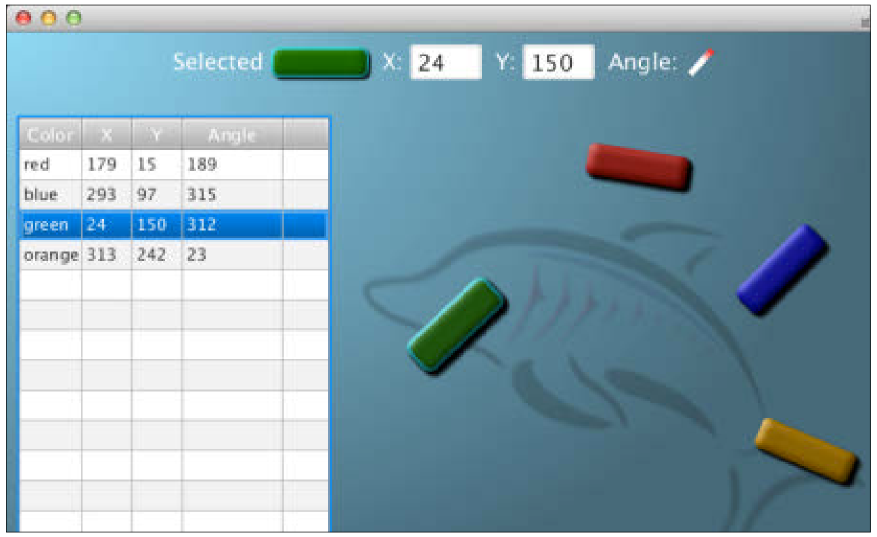
Doing Is Believing

OpenDolphin ships with many demos that serve as examples for every feature and capability. It is always helpful to start with a simple example to get your feet wet. To this end, we have created the DolphinJumpStart project on GitHub, which provides a standard project structure for either Maven or Gradle builds and has JavaFX views written in plain Java. (In other places, we take shortcuts and use GroovyFX.) The DolphinJumpStart project leads you through a series of seven steps from a simple JavaFX view up to a fully "dolphinized" view with client-server distribution.

Here is a sneak peek into the project. Let's assume we had a view with a JavaFX text field similar to this:

```
TextField field =
  new TextField();
```

Now, we would like to bind the content of this field to an OpenDolphin presentation model. We first need a model. Then we would like to retrieve the model using the name input, and we want to have a text attribute, as shown in **Listing 1**.



Color	X	Y	Angle
red	179	15	189
blue	293	97	315
green	24	150	312
orange	313	242	23

Demo 6: Push demo

LISTING 1 / LISTING 2 / LISTING 3

```
PresentationModel input = clientDolphin.presentationModel(
"input", new ClientAttribute("text"));
```

Download all listings in this issue as text

Once we have the presentation model, we can bind it against the view, as shown in **Listing 2**. Binding has the following effect: whenever the content of the text field changes, the presentation model is modified as well and the server is notified asynchronously.

Let's assume that we want to do something with the content on the server side, for example, store it in a database. For simplicity, we print the content to make sure it is available. The server-side action is shown in **Listing 3**.

You can trigger the server-side action by sending the following command:

```
clientDolphin.send("PrintText");
```

It's in the Binding

We have scratched only the surface of what's possible with the binding. OpenDolphin has many more pearls in that bag. You can bind against the dirty state of attributes such as enabled, visible, mandatory, label, tooltip, and so on. You can even do attribute-specific, client-side validation against a regular expression. We profit from JavaFX having very elaborate property binding and from OpenDolphin providing the information source.

But the killer OpenDolphin feature is *stable bindings*. Business applications are full of master-detail views. Sometimes they are obvious, but other times they are disguised as tabbed panes, internal frames, navigation bars, and so on. The push demo (see **Demo 6**) is a good example. The challenge is to get the consistent updates right without every view knowing every other view, a central "manager" who controls everybody, or excessive unbind and rebind activities. With OpenDolphin, the issue doesn't even arise. Every view knows its one and only presentation model, and this binding never changes.

Come in, We're Open

You are invited to join the OpenDolphin project. For an overview, you might want to look at the architecture section of the user guide. After the 1.0 release, the API will remain backward compatible.

There are many resources for OpenDolphin on the Web, including videos, screencasts, and technical presentations. OpenDolphin is open source with an Apache 2 license and is hosted on GitHub. We are happy to receive your questions, suggestions, and other contributions.

The first major applications built with OpenDolphin are now in production, and some are just about to go live. Early adopters have been a great source of inspiration, and have helped shape the library's functionality and performance. It was early adopters who discovered how useful the OpenDolphin approach is for automated testing. You get UI mocks and functionality mocks for free!

For functional testing, you simply create client presentation models, send a command, and validate the effect in the model store. To quote one developer, "This is UI testing without UI!"

MAKE IT EASY

Separating views makes it much easier to create, test, debug, and maintain them and—last but not least—adapt and extend them to meet changing requirements.

Conclusion

UI construction often appears to be the ugly stepchild of software engineering, and many applications invest little effort in the structure of their presentation code beyond vaguely referring to model-view-controller. OpenDolphin enforces a strict separation of views and a full decoupling of controllers with generic presentation models as the reliable single source of information. All communication happens asynchronously—automatically in the right thread.

Enterprise applications run on the server, and so does OpenDolphin. This opens endless opportunities for securely sharing data, live updates, multiuser-awareness, and collaborative working. Combine that with world-leading visualization technologies and you have OpenDolphin.

If you are about to embark on the JavaFX train, if you want to protect your investment in business logic against changing UI technologies, or if you otherwise seek enterprise readiness for your business applications, my best advice is to have a serious look at OpenDolphin.

OpenDolphin is small and lightweight enough to even run embedded on devices such as the Raspberry Pi. I am very confident that it is also ready for future mobile computing with JavaFX mobile just like it made its first promising appearances on the Web. `</article>`

LEARN MORE

- Twitter @OpenDolphin
- Dolphin reference documentation
- git@github.com:canoo/open-dolphin.git
- git@github.com:canoo/DolphinJumpStart.git
- Dierk König's videos on YouTube

SANDY RYZA

BIO

Learn MapReduce Programming by Analyzing Traffic Data

Write a program that processes a petabyte of data.

Want to write a program that processes a petabyte of data? Apache Hadoop is a platform—written in Java—that leverages clusters of commodity computers for storage and massively parallel processing of enormous data sets. The storage aspect is handled by Hadoop Distributed File System (HDFS), a distributed file system based on the Google File System (GFS). The parallel processing, which is the focus of this article, occurs in MapReduce, a programming model and implementation that executes code in a distributed fashion while abstracting the distributed systems details from the programmer. Hadoop's MapReduce API allows Java programmers to write programs that process, transform, and derive insights from petabyte-scale data sets.

IT SCALES

MapReduce allows a program that will work with five input records to scale up to **handle five trillion input records.**

The MapReduce Paradigm

The MapReduce programming paradigm is simple: it divides processing into two functions, each of which executes in parallel many times across a cluster on a subset of the data. The map function takes a key-value pair of input data and can have zero or more key-value pairs. After the map function has been called on each input record, the map outputs are sorted by key and sent to the reducers.The reduce function is called once for each unique map output key, with a list of the map output values corresponding to that key, and it can emit zeros or more key-value pairs of its own.

The Use Case

The California Department of Transportation (Caltrans) Performance and Measurement System (PeMS) provides detailed traffic data from sensors placed on freeways across the state, with updates arriving every 30 seconds. The Los Angeles area alone contains over 4,000 sensor stations covering all its freeways. While this is, frankly, truckloads of data, MapReduce allows us to leverage a cluster to process it in a reasonable amount of time.

We'll write a simple MapReduce program that computes average traffic volumes for each sensor station at each time of the week, providing a profile of a region's traffic load and the most heavily trafficked locations. This data can also act as a baseline for deeper analyses that look at additional statistics, such as standard deviation, or determine how traffic at specific times and places relates to the typical patterns.

The Input Data

While the data is available every 30 seconds, we don't need such fine granularity, so we will use the 5-minute summaries that PeMS also publishes. Thus, with 4,370 stations, we will be calculating 4,370 * (60 / 5) * 24 * 7 = 8,809,920 averages.

Each of our input data files contains the measurements for all the stations over a month. Each line contains a station ID, a time, some information about the station, and the measurements taken from that station during that time interval.

Listing 1 shows some example lines. The fields that are useful to us are the first, which tells the time; the second, which tells the station ID; and the tenth, which provides a normalized vehicle at that station at that time.

The Job

Our mappers will parse the input lines and emit a key-value pair for each line, where the key is an ID that combines the station ID with the time of the week, and the value is the number of cars that passed over that sensor during that time. Each call to the reduce function receives a station, a time of week, and the vehicle count values over all the weeks, and it computes their average. **Listing 2** shows what our mapper looks like.

The generic type arguments specify the input and output key-value classes for our mapper. You'll notice that the classes aren't String and the typical Java primitive wrappers such as Integer and Double. This is because key-value classes must implement the Writable interface, which specifies serialization methods that allow key-value pairs to be sent as bytes over the network.

We use Text for our output key, which will hold an ID that combines the time of week and sensor station ID. We use IntWritable for our output value, which will hold the traffic count. The input classes are in part dictated by the InputFormat that we will use, which, in this case, is TextInputFormat.

TextInputFormat breaks up the input files into lines and passes in each line as a record with a LongWritable key that holds the byte offset into the file and a Text value that holds the line. **Listing 3** shows what our reducer looks like.

Our reducer must take the map output types as its input types. It outputs the same keys, but with DoubleWritable averages attached to them.

Finally, we need to actually execute our program. To do this, we run a driver that sets up the job and submits it to the cluster. Because our code will be executed on many machines, we need to tell Hadoop what JAR file contains the code, so that Hadoop can distribute the code across the cluster. The setJarByClass method allows us to specify this JAR file by passing a class that resides in it (see **Listing 4**).

From here, MapReduce will handle splitting up our input data to send to the mappers; starting the mappers on machines across the cluster; telling reducers where they should get their mapper out-

INSIGHT FROM DATA

Hadoop's MapReduce API allows Java programmers to write programs that process, transform, and derive insights from petabyte-scale data sets.

LISTING 1 / LISTING 2 / LISTING 3 / LISTING 4

```
01/01/2012 00:00:00,312346,3,80,W,ML,.491,354,33,1128,.0209, 71.5
,0,0,0,0,0,0,260,.012,73.9,432,.0273,69.2,436,.0234,72.3,,,,,

,,,,,,,,,,
01/01/2012 00:00:00,312347,3,80,W,OR,,236,50,91,,,,,,,,,91

,,,,,,,,,,,,,,,,
01/01/2012 00:00:00,312382,3,99,N,ML,.357,0,0,629,.0155,67,0,0,0
0,0,0,330,.0159,69.9,299,.015,63.9,,,,,,,,,,,,,,,,,,
01/01/2012 00:00:00,312383,3,99,N,OR,,0,0,,,,,,,,,,,,,,,,,,,,,,,

,,,,,,,,,
01/01/2012 00:00:00,312386,3,99,N,ML,.42,0,0,1336,.0352,67.1,0,0,
0,0,0,0,439,.0309,70.4,494,.039,67.4,403,.0357,63.4,,,,,,,,,,,,,
,
```

Download all listings in this issue as text

put data; and, finally, writing the reducer outputs to files, usually on HDFS.

Passing Some Data on the Side

In addition to the input data that gets split up and sent out to our mappers, we might want to pass some job-specific configuration options to our map and reduce tasks on the side.

MapReduce makes this very simple for us. In our driver, we can set an arbitrary option on the configuration option passed to us, as shown in **Listing 5**. And we can pick up the setting in our mapper or reducer using the code shown in **Listing 6**.

SIMPLE PARADIGM

MapReduce divides processing into **two functions** that execute in parallel many times across a cluster on a subset of data.

Counters

Data in the real world is messy. Traffic sensor data, for example, often contains records with missing fields, and sensors in the field are bound to malfunction at times. While running our MapReduce job, it is often useful to count and collect metrics about what our job is doing.

For a program on a single computer, we might simply do this by adding in a count variable, incrementing it whenever our event of interest occurs, and printing out its value at the end. But when our code is running in a distributed fashion, aggregating these counts can be quite challenging.

Luckily, Hadoop provides a mechanism to handle this using *counters*. MapReduce contains a number of built-in counters that you will see in the console output upon completion of a MapReduce job.

```
Map-Reduce Framework
  Map input records=10
  Map output records=7
  Map output bytes=175
  ...
```

This information is also available in the Web UI, both per job and per task. To use our own counter, we can simply add a line like the one shown in **Listing 7** at the point in the code where the mapper comes across a record with a missing count.

Then, when our job is complete, we will see our count along with the built-in counters:

```
Averager Counters
  Missing vehicle flows=2329
```

LISTING 5 / LISTING 6 / LISTING 7 / LISTING 8

```
conf.set("my.made.up.configuration.option", "the.value");
```

Download all listings in this issue as text

It's often convenient to wrap your entire map or reduce function in a try/catch and increment a counter in the catch block, using the name of the exception class as the counter's name for a profile of what kind of errors come up.

Testing

Running a MapReduce program on a cluster, when we have access to one while writing and debugging our program, can take time. However, if we want to confirm that our basic logic works, we have no need for all the machinery that distributes and executes our code across a cluster.

Apache MRUnit is an open source project that makes writing JUnit tests for MapReduce programs about as easy as it could possibly be. Through it, we can test our mappers and reducers both separately and as a full flow. **Listing 8** shows a test that verifies that our mapper does what it should.

Conclusion

In this article, we used MapReduce to write an application that looks at the Caltrans PeMS freeway traffic data set to calculate the average traffic at each time of the week. We covered a couple of advanced features of the API that allow us to pass application-specific configuration options down to our tasks, as well as collect global statistics on their execution. Finally, we learned how to write unit tests for our MapReduce programs using MRUnit.

MapReduce allows a program that will work with five input records to scale up to handle five trillion input records. Hadoop's MapReduce API allows us to do this easily in pure Java.

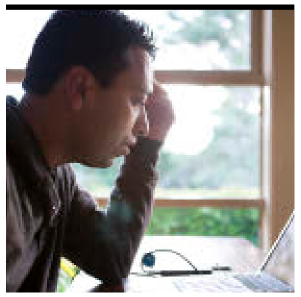
While MapReduce is very powerful, it can also feel low level, and it requires writing lines of code for common operations such as group-bys and joins. In addition, often a task requires a sequence of MapReduce jobs, where the outputs of one job become inputs to the next. The Apache Crunch project provides a higher-level Java library that greatly simplifies writing pipelines of MapReduce jobs. Crunch is oriented around tuples, making it much easier to work with and easier to pass around complex datatypes. `</article>`

HERE'S A HINT

The Apache Crunch project provides a higher-level Java library that greatly simplifies writing pipelines of MapReduce jobs.

LEARN MORE

- Google's original MapReduce paper: "MapReduce: Simplified Data Processing on Large Clusters"
- How to write, compile, and run a simple MapReduce job on Apache Hadoop
- "Apache Crunch: A Java Library for Easier MapReduce Programming"



VIKRAM GOYAL

BIO

Listen to author Vikram Goyal discuss pushing your Instagram applications to Facebook.

Part 2

Build Your Own Instagram Java ME Application

Learn how to post your images on Facebook with Java ME.

In the first part of this two-part series, I showed how to modify your images using nothing but good old Java ME image manipulation. If you need a refresher, here is a link to Part 1.

In this part, I show how to post a message to Facebook so that you can present your beautiful images to the world. Posting to Facebook is a very difficult proposition that required me to jump through several hoops.

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

Facebook Integration

There is no officially supported Facebook API for Java ME devices. In fact, I could not find one *unofficially* supported API that worked. So I had to build a very rudimentary one that would perform the job at hand.

It doesn't help that Facebook constantly changes its policies, access, and authentication mechanisms. For one of the most popular companies in the world, it has terrible developer support.

The final process involved several workarounds. But once the workarounds were settled on, putting the pieces together was a piece of cake. The final outcome was surprisingly robust and worked on both the emulator (Java ME SDK 3.0.5) and the target device (Nokia N95).

This integration required several steps, which I will list in order.

NOT SO EASY

Posting to Facebook is a very difficult proposition that required me to jump through several hoops.

Step 1: Create a Dummy Facebook User

One of the hardest jobs in developing an application that can integrate with Facebook is authentication and access control. Constantly having to authenticate and provide access to the users' news feeds and publish streams is difficult.

In addition, on older phones and emulators, it is difficult to keep inserting the users' credentials each time we post an image to their photo stream.

To overcome this, I have created a generic user for the purposes of this article and called it PhotoShare .AppArticle. It can be accessed here.

You might want to use this dummy user or create your own. If you create your own, you will need a separate access control token (described later).

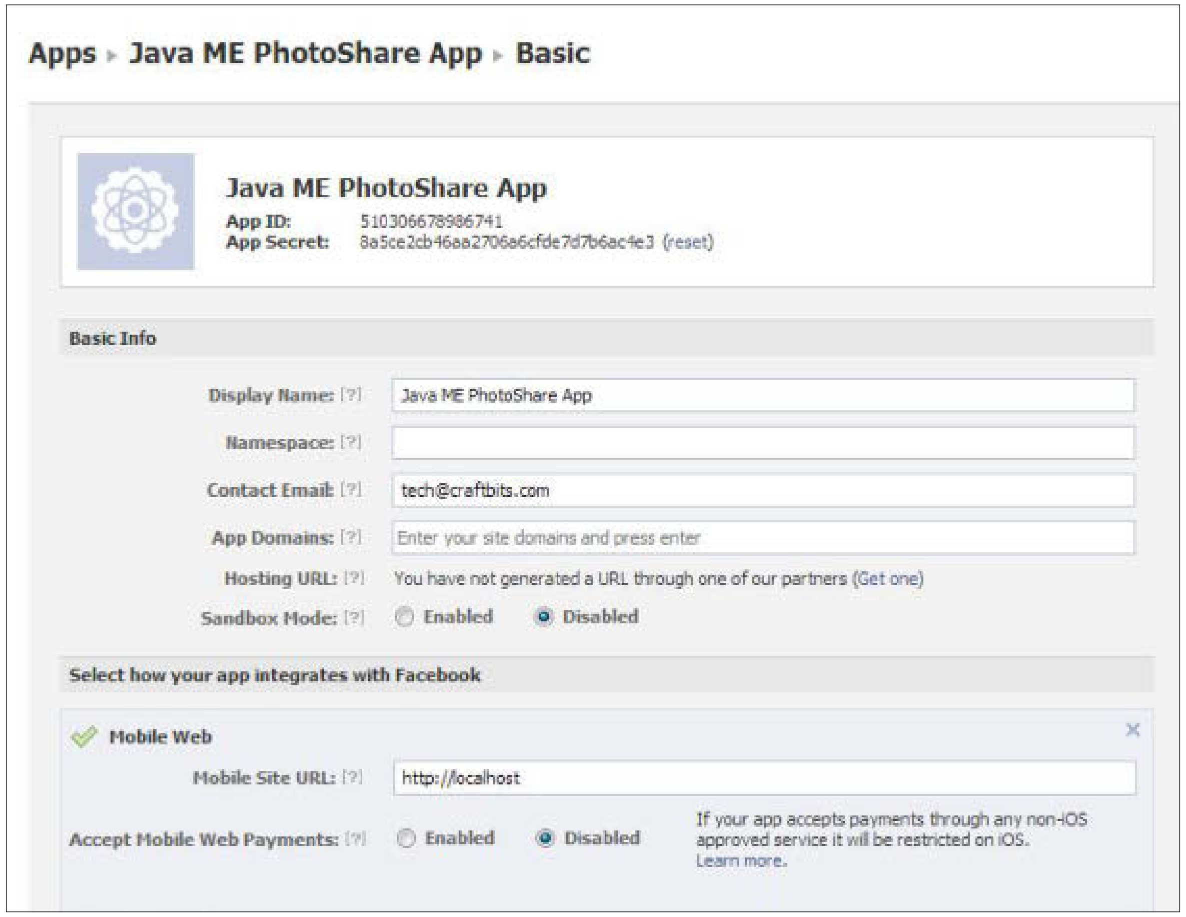
Step 2: Tell Facebook About Our PhotoShare Application

Before our PhotoShare application can post Instagram-style images to the dummy user account that I created in Step 1, Facebook needs to know about this application.

In addition, the dummy user needs to grant access to this application so that it can publish to the photo stream.

To tell Facebook about our PhotoShare application, I went to http://developers .facebook.com and created a new application that I called Java ME PhotoShare App and tagged as a mobile Web application. For the mobile

PHOTOGRAPH BY JONATHAN WOOD/ GETTY IMAGES



Apps ▸ Java ME PhotoShare App ▸ Basic

Java ME PhotoShare App
App ID: 510306678986741
App Secret: 8a5ce2cb46aa2706a6cfde7d7b6ac4e3 (reset)

Basic Info

Display Name: [?]	Java ME PhotoShare App
Namespace: [?]	
Contact Email: [?]	tech@craftbits.com
App Domains: [?]	Enter your site domains and press enter
Hosting URL: [?]	You have not generated a URL through one of our partners (Get one)
Sandbox Mode: [?]	Enabled [✓ Disabled]

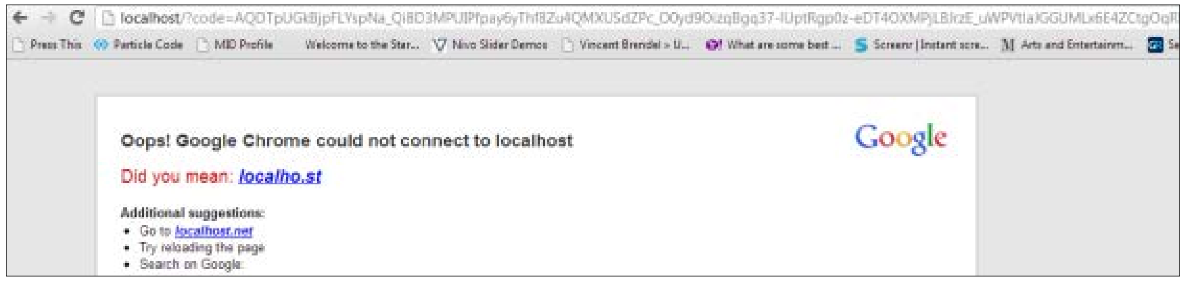
Select how your app integrates with Facebook

✓ Mobile Web

Mobile Site URL: [?]	http://localhost
Accept Mobile Web Payments: [?]	Enabled [✓ Disabled]

If your app accepts payments through any non-iOS approved service it will be restricted on iOS. Learn more.

Figure 1



localhost/?code=AQDTpUGkBjpFLYspNa_Qi8D3MPUIPfpay6yThf8Zu4QMXUSdZPc_O0yd9OizqBgq37-lUptRgp0z-eDT4OXMPjLBlrzE_uWPVtlaXGGUMLx6E4ZCtgOqR

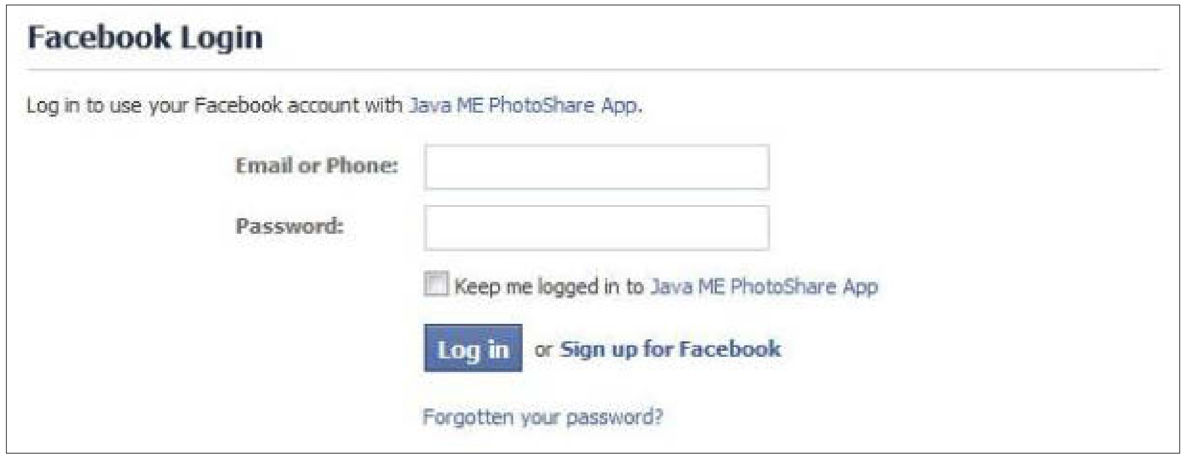
Oops! Google Chrome could not connect to localhost

Did you mean: *localho.st*

Additional suggestions:

- Go to *localhost.net*
- Try reloading the page
- Search on Google:

Figure 2



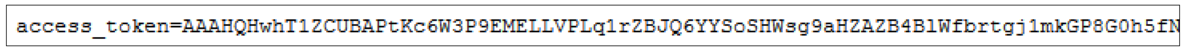
Facebook Login

Log in to use your Facebook account with Java ME PhotoShare App.

Email or Phone:	
Password:	

☐ Keep me logged in to Java ME PhotoShare App

Figure 3



```
access_token=AAAHQHwhT1ZCUBAPtKc6W3P9EMELLVPLq1rZBJQ6YYSoSHWsg9aHZAZB4BlWfbrtgj1mkGP8G0h5fN
```

Figure 4

site URL, I put in http://localhost. See **Figure 1**.

Step 3: Connect the Dummy User and Our Application

We have a dummy user, and we have notified Facebook about our app. We now need to let Facebook know that our PhotoShare app is allowed to post images to the dummy user's Facebook profile. To do this, the dummy user needs to grant an access token to our app. This magical access token is all we need to connect these two. Most access tokens are short-lived (around a one- to two-hour duration), but you can get them extended (one to two years).

Thankfully, it is very easy to get an access token by following these steps:

Step 3A: Log in and associate the app and your account (or the dummy account). In a browser window, navigate to here.

If you're already logged in to Facebook from your own account in another browser tab, the message shown in **Figure 2** appears.

If you're *not* logged in, the login page shown in **Figure 3** appears.

For the purposes of this exercise, use the dummy account that we created in Step 1. The username for that account is ps_app@hotmail.com, and the password is photo_share.

Once you log in successfully, the browser will redirect to localhost, as we saw in **Figure 2**. You need to copy the code that Facebook returns as part of that URL (see the area where you type in the URL). You have now associated your (or the dummy Facebook user) account with the PhotoShare App.

You still need an access token.

Step 3B: Get the access token, or use the one that I already created. To get a short-lived access token, all you need to do is type (all as one line) into a browser the information shown in **Listing 1**.

You will get a simple response back from Facebook with your access token (see **Figure 4**).

Notice that the access token has an extra parameter that specifies when it will expire. It indicates the number of seconds you have to use the token before it will no longer be valid. Because ours is a mobile application, it automatically receives a validity of 60 days.

If this step gets to be too much for you to handle, just use the access token that I already created (see **Listing 2**).

This token grants our PhotoShare App access to the publishing stream of the dummy user created in Step 1. In the next few steps, I will use this access token to publish the Instagram-style photos that I created in Part 1 of this series.

DIY FACEBOOK API

There is no officially supported Facebook API for Java ME devices. In fact, I could not find one unofficially supported API that worked. So I had to build a rudimentary one.

Step 4: Use the Access Token to Post to Facebook

Now we come to the non-Facebook steps.

You might recall from Part 1 of this series that we have a transformed image being displayed to the end user. When the user decides to select **Publish**, we need to post that transformed image to Facebook. We can do that now, using the access token we got in the previous step, as shown in **Listing 3**.

When the user selects **Publish**, the following steps take place.

Step 4A: Use a JPEG encoder to encode the image. The image we have at the moment is not suitable for transfer over a POST request to Facebook. We need to encode it using a JPEG encoder. I have used a third-party library to encode the image.

Step 4B: Create the POST request. I have created a separate class called the NetworkConnector for the actual bundling of the data and for simulating a POST request to Facebook. The NetworkConnector class creates a boundary for all the parameters and specifies the Content-Type of the request as multipart/form-data. It then sends the data via the normal HTTPConnection class.

Step 4C: Send the data. So far, I haven't said anything about where the POST request is to be sent. Here is a special Facebook Graph API URL for posting to the dummy user's account: https://graph.facebook.com/me/photos.

Coupled with the access_token parameter and the media data, this URL results in the transformed image being posted on the dummy user's news feed, as seen in **Figure 5**.

LISTING 1 LISTING 2 LISTING 3

```
https://graph.facebook.com/oauth/access_token?client_id=510306678
986741&redirect_uri=http://localhost&client_secret=8a5ce2cb46aa27
06a6cfde7d7b6ac4e3&code=<INSERT CODE FROM STEP 3A HERE>
```

Download all listings in this issue as text

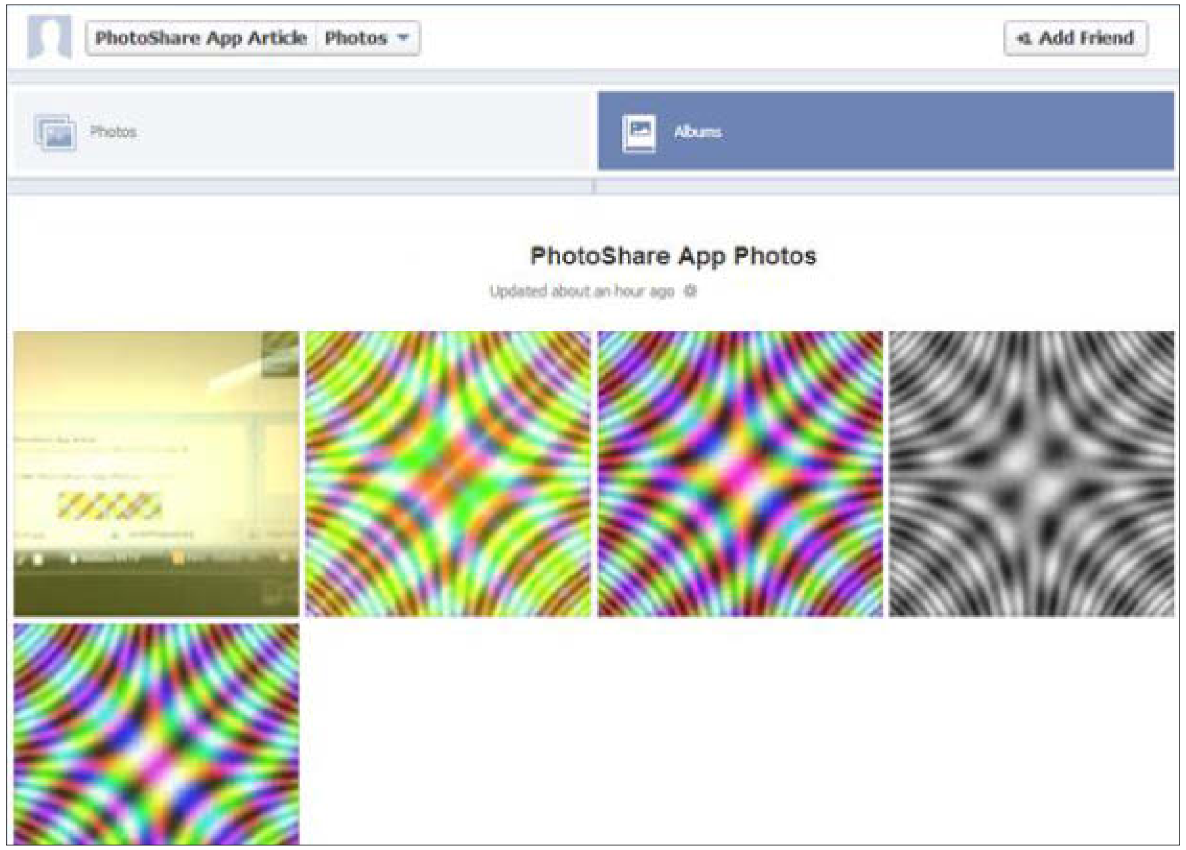


Figure 5

And that's it! We have successfully created an Instagram-type app on a Java ME–enabled phone using nothing but traditional Java ME APIs.

Conclusion

This is the second and final part of my Instagram-like app development in Java ME. This two-part series demonstrated how Java ME is capable of creating advanced applications that can rival traditional smartphone apps.

In the first part, I created the basic filters—vintage and grayscale—and implemented them using normal Java ME routines. In Part 2, I showed how to post the modified images to Facebook using Facebook's Graph API. The example app is not perfect, but it shows the key points and could form the basis of a more elaborate and user-friendly application. </article>

LEARN MORE

- [Facebook's Graph API]()
- [How to use the Graph API to upload photos to a user's profile]()

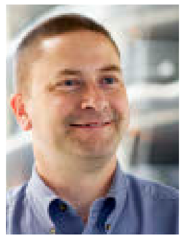
3 Billion Devices Run Java

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

Java™ | #1 Development Platform

ORACLE®

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



In the May/June 2013 issue, Michal Lisiecki gave us a code challenge around parsing dates with SimpleDateFormat. He gave us a code fragment and its result and asked for a fix. The correct answer is #2. The DateFormat class (the superclass of SimpleDateFormat) has a lenient property that in our case should be set to false in order to force strict pattern matching. Be careful when using this class.

The correct code looks like this:

```
SimpleDateFormat dateFormat = new SimpleDateFormat(date
FormatString);
dateFormat.setLenient(false);"
```

This issue's code challenge comes from Simon Ritter, a Java evangelist at Oracle, who presents us with a JavaFX problem.

1 THE PROBLEM

We want to use the SplitPane control in JavaFX to allow the user to change the size ratio of two parts of the user interface. The issue is how to add nodes so that each side of the divider in the SplitPane can indeed be resized.

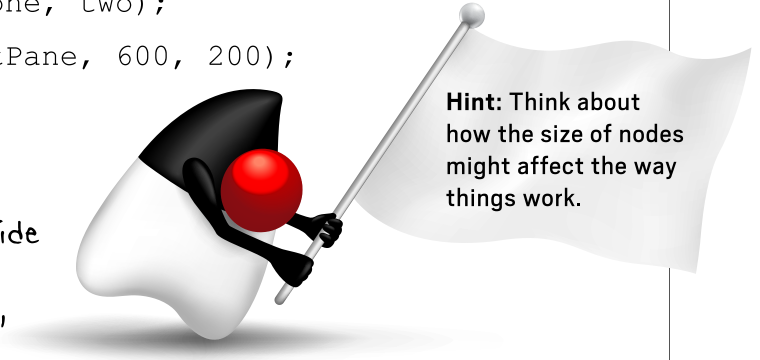
2 THE CODE

Consider the following start method for a simple JavaFX application that tries to place a button on each side of the divider in a SplitPane:

```
  @Override
  public void start(Stage stage) {
    SplitPane splitPane = new SplitPane();
    Button one = new Button("Left Button");
    Button two = new Button("Right Button");
    splitPane.getItems().addAll(one, two);

    Scene scene = new Scene(splitPane, 600, 200);
    stage.setScene(scene);
    stage.show();
  }
```

When this application is run, the left side of the divider will be filled with the "Left Button" and the "Right Button" will be immediately to the right of the divider. Trying to drag the divider in either direction has no effect.


Hint: Think about how the size of nodes might affect the way things work.

3 WHAT'S THE FIX?

1) Rather than adding both buttons using the addAll() method of ObservableList<Node>, it is necessary to add them sequentially using two separate calls to the add() method.

2) Add a call to requestLayout() on SplitPane to force the layout to be recalculated correctly.

3) Place each Button object in a separate Pane and add the panes instead of the buttons to the SplitPane:

```
Pane left = new Pane();
left.getChildren().add(one);
Pane right = new Pane();
right.getChildren().add(two);
stackPane.getItems().addAll(left, right);
```

4) Set the divider position explicitly to be in the middle using a call to setDividerPosition:

```
splitPane.setDividerPosition(0.5);
```

GOT THE ANSWER?

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

PHOTOGRAPH BY BOB ADLER,
ART BY I-HUA CHEN