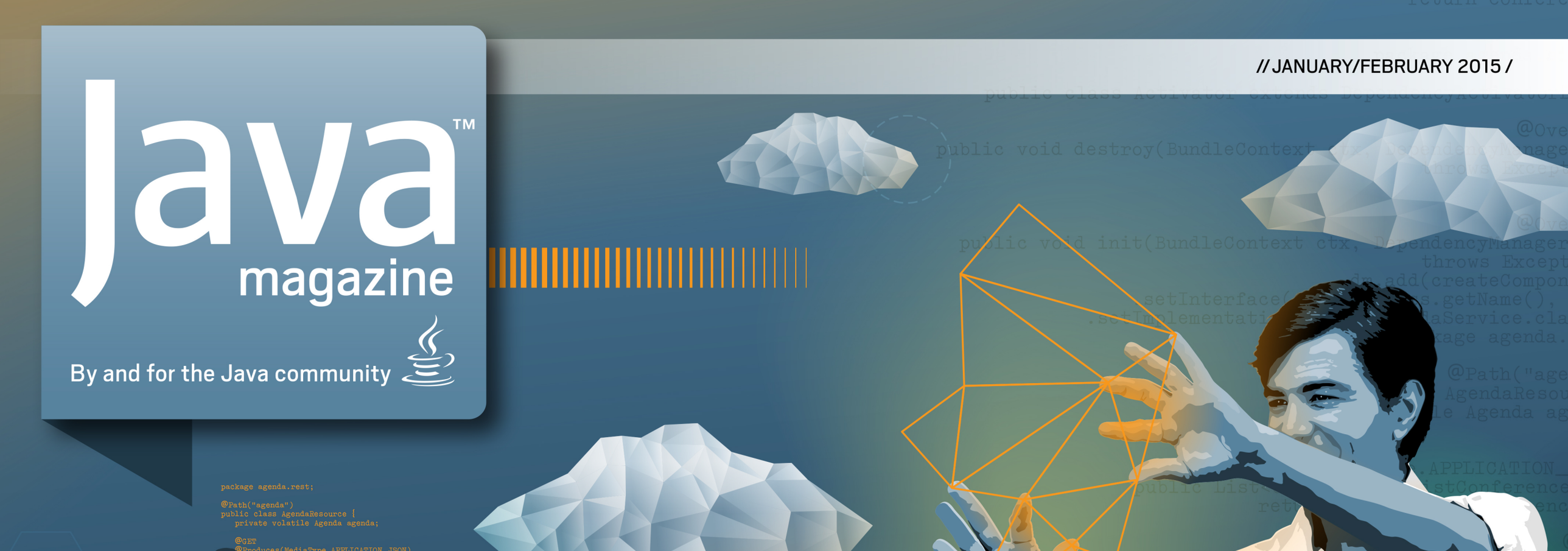
// JANUARY/FEBRUARY 2015 /

Java™ magazine

By and for the Java community



ORACLE CLOUD

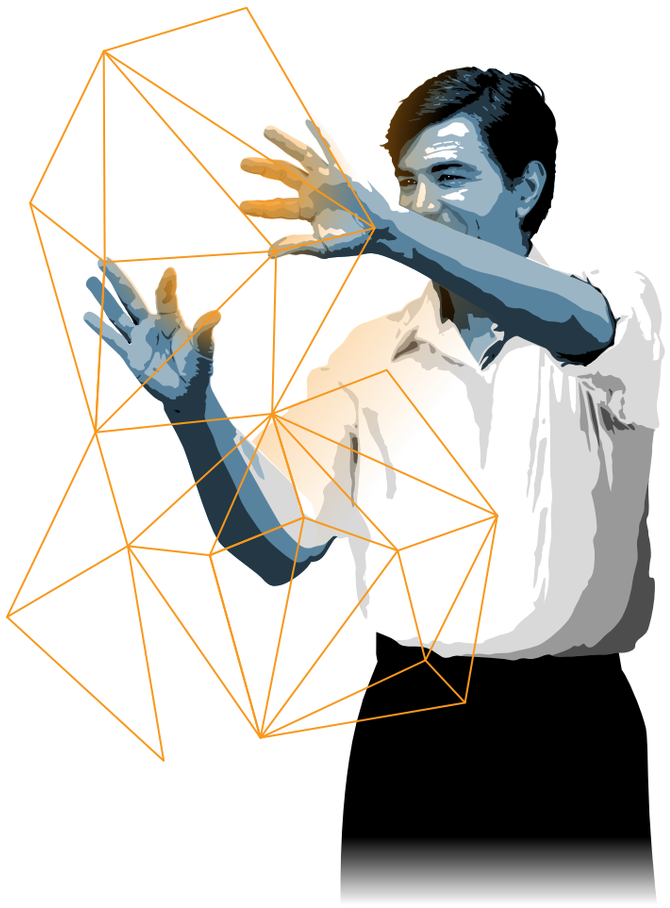
Platform for Innovation

Java-based services provide flexible tools for developing and deploying new applications in the cloud

ORACLE.COM/JAVAMAGAZINE

ORACLE

//table of contents /



EDITORIAL

Editor in Chief
Caroline Kvitka

Community Editor
Yolande Poirier

Java in Action Editor
Michelle Kovac

Technology Editor
Tori Wieldt

Contributing Writer
Kevin Farnham

Contributing Editors
Claire Breen, Blair Campbell, Kay Keppler, Karen Perkins

DESIGN

Senior Creative Director
Francisco G Delgadillo

Senior Design Director
Suemi Lam

Design Director
Richard Merchán

Contributing Designers
Jaime Ferrand, Diane Murray, Arianna Pucherelli

Production Designers
Sheila Brennan, Kathy Cygnarowicz

PUBLISHING

Publisher
Jennifer Hamilton +1.650.506.3794

Associate Publisher and Audience Development Director
Karin Kinnear +1.650.506.1985

ADVERTISING SALES

President, Sprocket Media
Kyle Walkenhorst +1.323.340.8585

Western and Central US, LAD, and Canada, Sprocket Media
Tom Cometa +1.510.339.2403

Eastern US and EMEA/APAC, Sprocket Media
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 (US/Canada)

Oracle Services
+1.888.283.0591 (US)

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

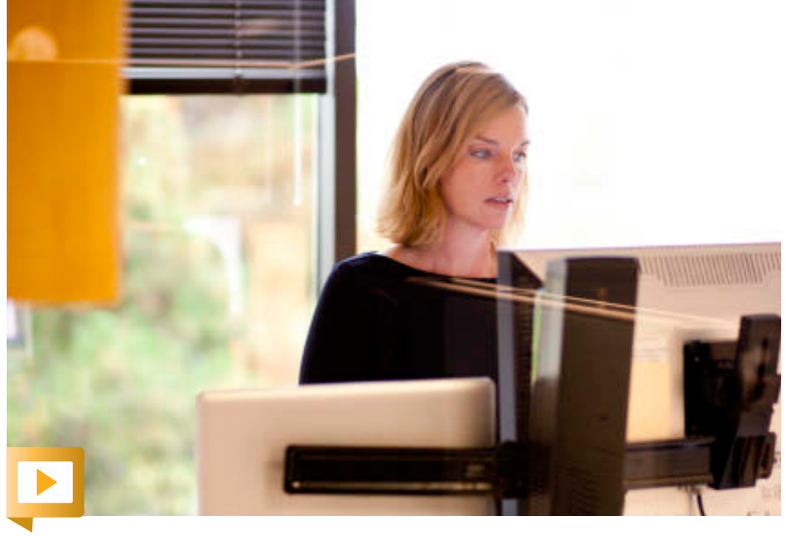
java@halldata.com **Phone** +1.847.763.9635

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 GTxcel



Tools make the cloud. To support modern development, today's cloud development environments must support the full software development lifecycle. Developers working in the cloud need tools for builds, continuous integration, source control, and team collaboration. With those tools at the ready, they can take advantage of the cloud as a platform for innovation.

Cloud Computing

In our interview with Mike Lehmann, vice president of product management for Oracle Cloud Application Foundation, we explore Oracle's Java-based cloud services for developers, and the benefits they offer. "Taking a new business idea and building an application to support it can be done much more quickly and with lower risk than in the past," says Lehmann. Read the interview for his insights on Oracle Java Cloud Service, Oracle Developer Cloud Service, and other services that give developers an end-to-end Java development and deployment environment.

Ready to test out these services? Don't miss Hardshad Oak's article, "Get Started with Oracle Developer Cloud Service." Plus, Bert Ertman shows you how to build modular cloud applications in Java.

It's a new year, and a time when people traditionally think about self-improvement. If you are looking to raise your job prospects and expand your network, Bruno Souza and Edson Yanaga have just the plan for you, and it includes code, community, and—yes—cloud. Their action items will help you to raise your possibilities for the future.

Finally, this will be my last issue as editor in chief of *Java Magazine*. I want to thank you, the readers, for your enthusiastic support of the magazine. Lots of great content is in the works for 2015, so please keep reading (and coding).

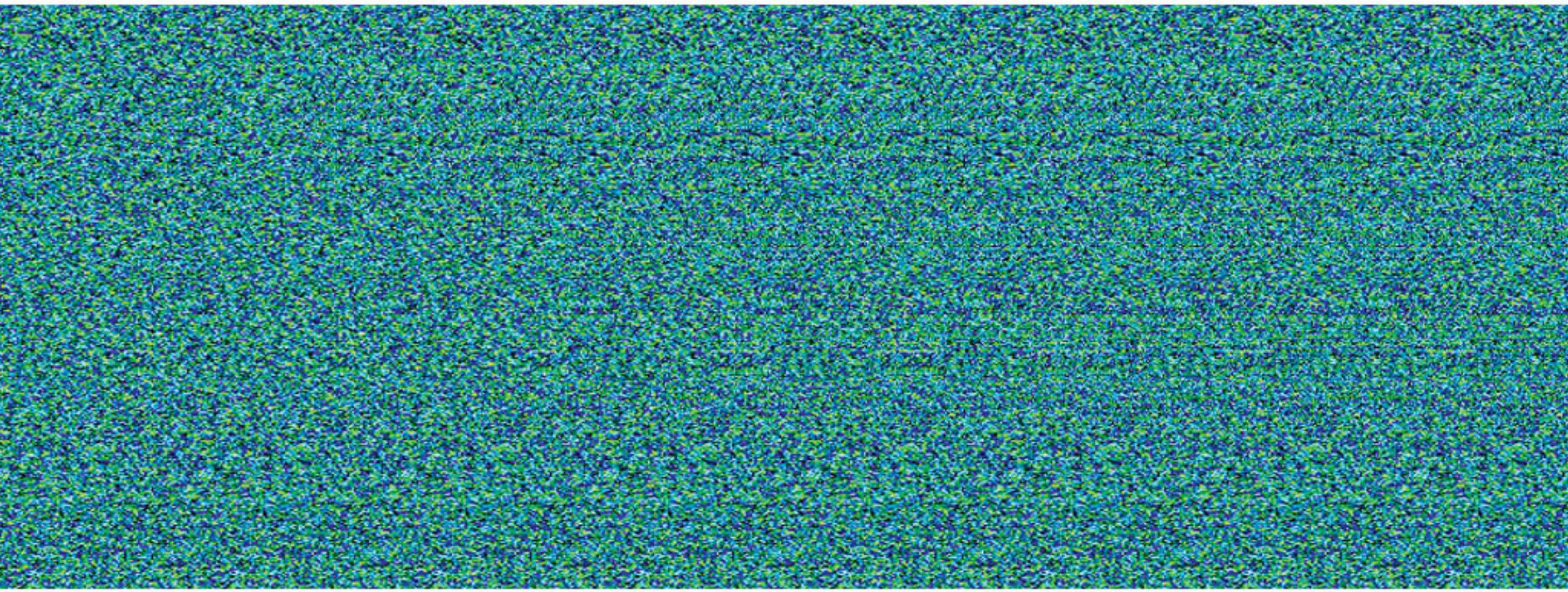
Caroline Kvitka, Editor in Chief BIO

//send us your feedback /

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

PHOTOGRAPH BY BOB ADLER

The answer is right in front of you



Java Image Enabling SDKs that Help You See the Big Picture

At first glance it may seem difficult, but it's really quite simple. Atalasoft's JoltImage product is a proven SDK for image enabling your Java-based web applications, easily. Image enabling helps to add dimension to your data, so you can uncover insights such as correlations and causations hidden inside your 2-dimensional documents. Our SDK does the heavy lifting for you, saving time, money, and the headaches of figuring it out yourself. Backed by our highly knowledgeable & caffeinated support engineers, JoltImage will enable your success and make the big picture so much easier to see.

Atalasoft JoltImage

Click for tips on viewing the stereogram

Image enabling experts & bacon connoisseurs. Visit us online to see our full line of SDK products for .NET and Java

Atalasoft
A Kofax Company
www.atalasoft.com

PLATFORM FOR INNOVATION

The Oracle Cloud platform's Java-based services provide flexible tools for developing and deploying new applications.

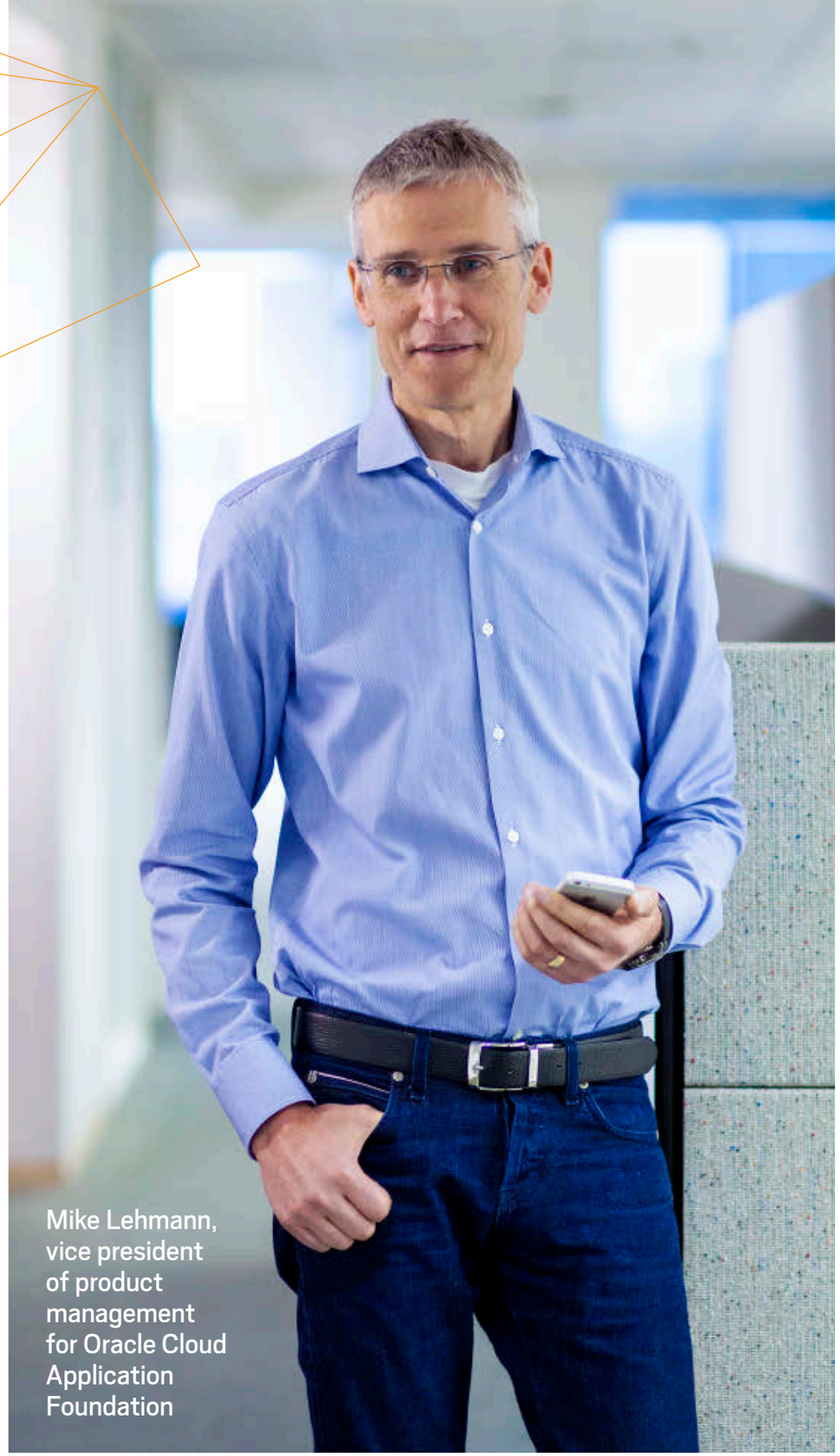
BY CAROLINE KVITKA

Cloud Computing

Cloud computing presents a growing number of opportunities and services for developers, especially in terms of simplicity, productivity, and innovation. We sat down with Mike Lehmann, vice president of product management for Oracle Cloud Application Foundation, to talk about what's new in the Oracle Cloud platform for developers.

Java Magazine: What does the rise of cloud computing mean for innovation, cost savings, and productivity for Java developers?

Lehmann: The emergence of the cloud is changing how developers and administrators work with software and hardware. Effectively what you're able to get now in a dramatically simplified fashion is instant access to compute capacity and enterprise software in the cloud. That changes the equation for developers and companies building applications. Taking a new business idea and building an application to sup-



Mike Lehmann, vice president of product management for Oracle Cloud Application Foundation

ART BY I-HUA CHEN; PHOTOGRAPHY BY BOB ADLER/GETTY IMAGES

Anand Kothari, director of product management at Oracle, and Lehmann work on cloud architecture documents.

THE FLEXIBILITY TO INNOVATE

"You have a true end-to-end Java development and deployment environment with a huge amount of flexibility and opportunity to build innovative applications."

port it can be done much more quickly and with lower risk than in the past.

The cloud is also changing how developers architect applications. For example, suppose I want to build a Java application with a mobile front end, use caching for performance and queuing services for integration, and include a standard SQL back end for persistence. Already today, that is simply a collection of services available in Oracle Cloud for developers to use as needed. It is a next-generation paradigm for what we used to do in what historically was called middleware on premises. With Oracle Cloud, developers can focus on building and deploying rather than deciding on platform infrastructure and trying to get it installed and set up.

Java Magazine: What Oracle Cloud services are available for developers?

Lehmann: Our Java-based cloud offerings start with Oracle Java Cloud Service, which enables developers to get a Java EE server provisioned in minutes. The resulting server is a user-defined cluster of Java infrastructure, with lifecycle management cloud tools that enable backups, recovery, patching, scale-in, and scale-out. The underlying infrastructure is Oracle WebLogic Server, our standards-based, market-leading application server. And you can plug directly into Oracle Database Cloud Service for persistence.

Oracle Developer Cloud Service, which works hand in hand with Oracle Java Cloud Service, provides a full development lifecycle platform. It includes a Git repository and a Hudson build system, for source code management to support a continuous integration approach. It's integrated with standard development tools such as Eclipse and Oracle JDeveloper, and it features collaboration tools including a built-in wiki, issue tracking, and team management.

We are also complementing Oracle Java Cloud Service with several core platform services.

First, Oracle Messaging Cloud Service enables you to build messaging applications based on JMS [Java Message Service] queues and topics in the cloud. It can be used in client applications through a REST API as well as a native Java API.

Next, the Oracle Coherence caching capability caches Java objects in memory. This is a new extension of Oracle

Java Cloud Service using a capability of Oracle WebLogic Server called Managed Coherence Servers and will be available in 2015.

In 2015, we'll also have Oracle Java SE Cloud Service, which is the Java Virtual Machine [JVM] in the cloud, because many of our customers want to build standard JVM-based applications. They might be using a popular Java framework such as Spring on Java, or they might even be using one of the popular emerging Java-based scripting languages such as Play. Oracle Java SE Cloud Service, like Oracle Java Cloud Service, enables direct integration into the large Oracle Cloud ecosystem with Oracle Developer Cloud Service, Oracle Java Cloud Service, Oracle Database Cloud Service, and other services.

We are also very aware of the rise of server-side JavaScript solutions such as Node.js and so, in parallel with Oracle Java SE Cloud Service, we are building out Oracle Node Cloud Service, for availability in 2015. This will bring a first-class JavaScript server and, with Oracle Developer Cloud Service, a JavaScript development solution to Oracle Cloud, making it easy to build and deploy such solutions paired with our Java cloud services and integrated with our database cloud service.

As you can see, even without going into the higher-level services for mobility, integration, and document and business intelligence, you have a true end-to-end Java development and deployment environment with a huge amount of flexibility and opportunity to build innovative applications.

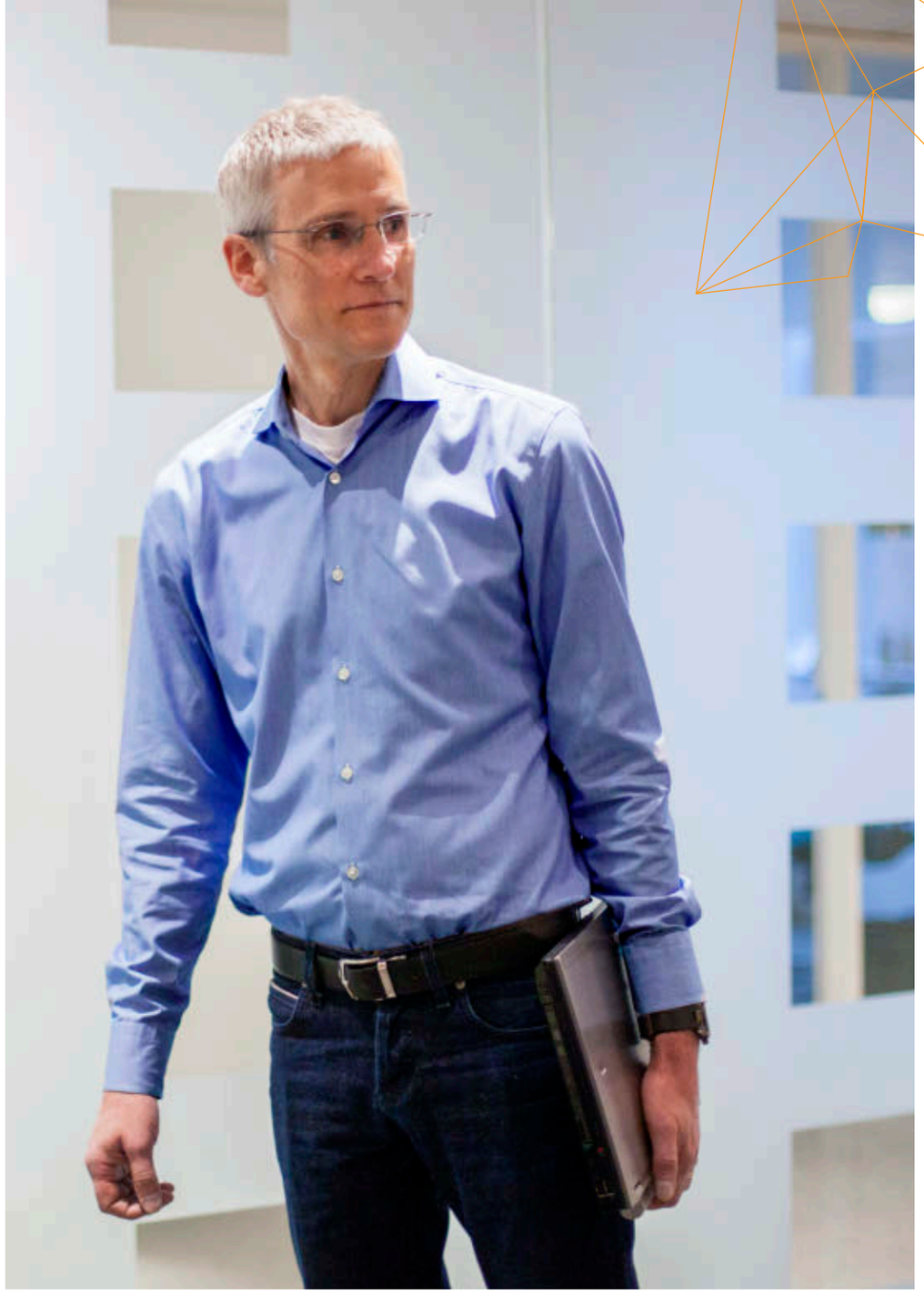
Java Magazine: Would you say that Oracle is trying to give developers everything they need for the way that they want to work?

Lehmann: We're building out the full application lifecycle management infrastructure with Oracle Developer Cloud Service, to answer the questions "How do I build?," "How do I manage the source code?," and "How do I manage my development team?" On the deployment side, with our Java cloud services, we're providing a very rich,

MODERN THINKING

"[The cloud] definitely is changing how people construct applications because they have an integrated team environment that can provision a test environment incredibly fast."

Nilesh Junnarkar, director of software development at Oracle, talks with Lehmann about load balancer provisioning.



Oracle is building out a full application development lifecycle management infrastructure with Oracle Developer Cloud Service, says Lehmann.

multitiered topology that you typically have in an application but doing it in an almost transparent way. When you ask for a multinode Oracle Java Cloud Service cluster for high availability and higher throughput, what you also typically get is a software load balancer in front of this environment. If you've chosen to add in a caching layer, then you'll get a multinode Oracle Coherence cluster. And finally, you're backed by an Oracle database. So for a developer you get both sides—team development; continuous integration; and a full-fledged, production-quality deployment infrastructure.

A classic example that you would use even in that scenario is you might have a layer in front of your Java application layer, perhaps using a Node.js front end where you're shaping some of the JSON objects that you want for your front-end HTML5 applications. Architecturally, without a lot of effort, you can bring these multitier architectures together in interesting ways that previously would have taken weeks or maybe even months, depending on the complexity of your IT environment. Now it's basically on demand. So it really changes how you think about your applications and how you architect your applications. You're getting an enterprise-quality solution out of the box from Oracle.

Java Magazine: How do these services support modern development?

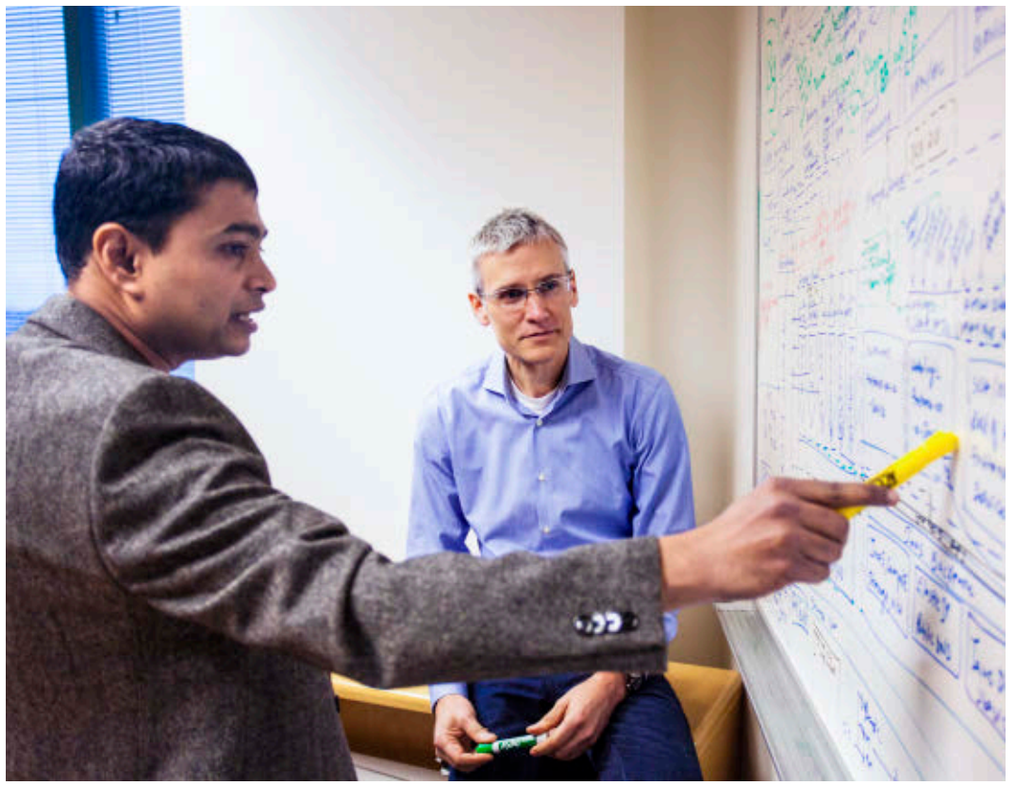
Lehmann: The use of build integrations systems such as Hudson, combined with Maven for dependency management and Git for source code management, has been an emerging set of trends and styles for doing development for some time. It's giving a continuous integration solution out of the box and enabling you to even consider continuous delivery models.

What is a little bit different now in the cloud is that previously you had to set that up. And there was complexity with it. In the cloud, we give you that environment out of the box. It definitely is changing how people construct applications, because they have an integrated team environment that can provision a test environment incredibly fast—or, more likely, in a test environment they can provision, test, and tear down very quickly. These capabilities are simply part of the environment versus being something you spend a huge amount of investment setting up. It truly is a modern development paradigm.

Java Magazine: Are people more comfortable these days moving to public cloud?

Lehmann: There has been the sort of ongoing debate around public cloud versus private cloud, and my sense is that this debate will slowly go away. I can see already in our customers when we talk about the cloud that more and more people are becoming very comfortable with moving development and test to the cloud, but keeping their

Kothari and Lehmann work on an Oracle Java Cloud Service component diagram.



production workloads on premises. I think that's just part of the shift. Two years ago, they weren't going to do anything in the public cloud. Now, it's dev and test. In another couple of years, as people start trusting and believing that these are secure, scalable, enterprise-class environments, they'll start moving production workloads.

Java Magazine: Why is Java the best language and platform for the cloud?

Lehmann: Java is effectively a community standard. The Java language has the Java Community Process around it for standardizing it and managing the overall technical standards. There is industry consensus around Java, that it is the standard programming language and programming environment that most people use in the cloud. That is complemented by a competitive server-side industry. So as a customer, you have both innovation and a competitive marketplace. And that simply carries over into the cloud. If you look at most of the cloud environments, the vast majority of them are building infrastructure on Java because that is the *lingua franca* of most developers today.

Java Magazine: What makes Oracle's cloud platform unique?

Lehmann: I think the big differentiator is that we bring two things together. First, we are the steward of Java so we are fairly aggressively bringing the most-recent releases of the Java platform into the cloud.

Second, one of the cool things that's different about what Oracle does in the cloud is that we are not only taking enterprise-class software to the cloud that our customers run mission-critical businesses on, but we're making doing that super simple in the cloud.

Java Magazine: What's your vision for cloud development?

Lehmann: I think it's an exciting change how all these services can be so easily used together to build new types of applications that simply weren't possible or were enormously complex in the past. When you start bringing in big data, social, and mobile services to the cloud as standard first-class, easily accessible services, it gets even more interesting. Rather than building applications centered around one service, modern applications will be built on top of a collection of cloud services in standard architectural patterns. And to me this new style of application development and application architecture based on patterns of services working together is going to rapidly emerge as the most exciting space to be in for the future. Very different and unique applications doing things that have never been done in the past will emerge out of this. I know I am pretty excited to participate and watch it happen! `</article>`

MORE ON TOPIC:

Cloud Computing

LEARN MORE

- Oracle Java Cloud Service
- Oracle Developer Cloud Service

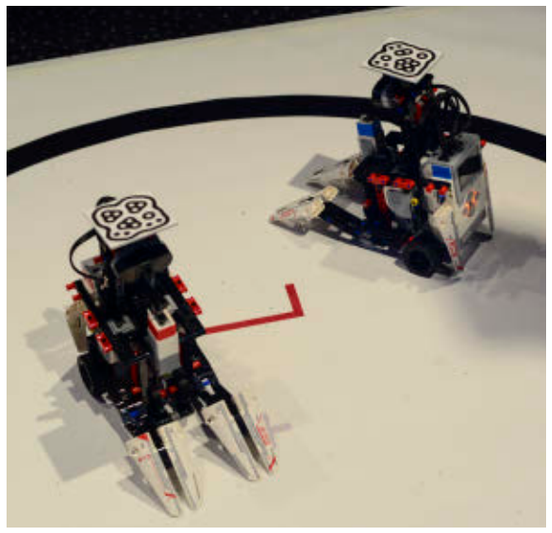
JetBRAINS

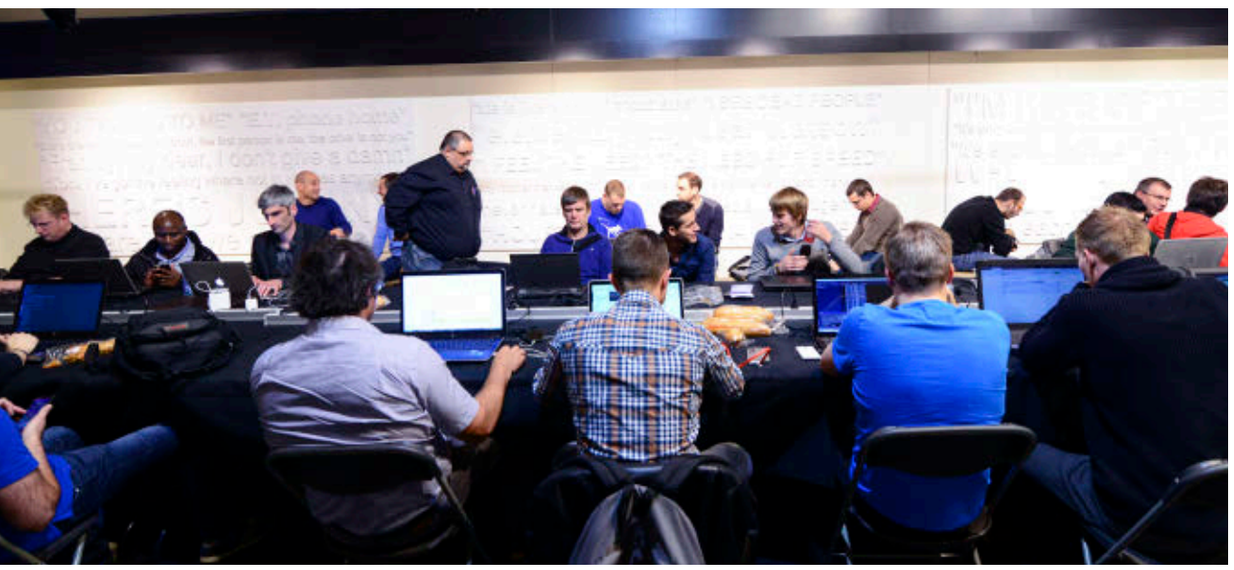
TURN PRO with IntelliJ IDEA

The Most Intelligent Java IDE

Download

*Warning: There is no going back





Clockwise from top left: Oracle's Brian Goetz gives a conference talk, Lego Mindstorms, Oracle's Richard Bair and Jasper Potts demo the telematics car, hackers at the Hackergarten.

DEVOXX DELIVERS

PHOTOGRAPHS COURTESY OF BEJUG

At Devoxx, conference attendees pondered the "infinite possibilities" of Java SE 8 and the upcoming Java EE 8. Devoxx, Europe's largest developer conference, took place in Antwerp, Belgium, November 10–14, 2014. The conference (which sold out two months in advance) attracted 3,500 attendees from 44 countries and 180 speakers.

New additions to Devoxx included a playground, Ignite talks, and a Devoxx Hunt game.

In the playground, developers showed off projects and programmed new ones. Some of the projects included Makeblock and Lego Mindstorm robots,

Top: Devoxx attendees relax and work. Bottom: Hackergarten updates.

Raspberry Pi's with Java 8, and 3-D modeling and printing.

More than 20 five-minute Ignite talks, on a variety of technical and nontechnical topics that speakers are passionate about, were held during lunch breaks at Devoxx.

Devoxx Hunt brought gamification to the conference. Attendees used an app to search for 160 iBeacons that were hidden throughout the venue and in Antwerp. Players received points for every beacon they found and unlocked, and could track their progress on a leader board. The idea caught on quickly, says Devoxx Managing Director **Stephan Janssen**, who reported that attendees were running around with their phones out, bumping into each other, and even missing sessions to play Devoxx Hunt.

Stephan Janssen discusses Devoxx highlights.

The Hackergarten was back at Devoxx this year, giving developers a chance to contribute to open source projects. They could write a plugin, a tutorial, or small features, or fix bugs for a project that interested them.

The conference also included one-hour conference talks, birds-of-a-feather (BOF) sessions, 15-minute quickie sessions, a two-day university program, and "tools in action" sessions that focused on technical tools and APIs. In addition, new this year was a series of startup sessions focused on lean startups, venture capital, angel investors, and trends.

PHOTOGRAPHS COURTESY OF BEJUG

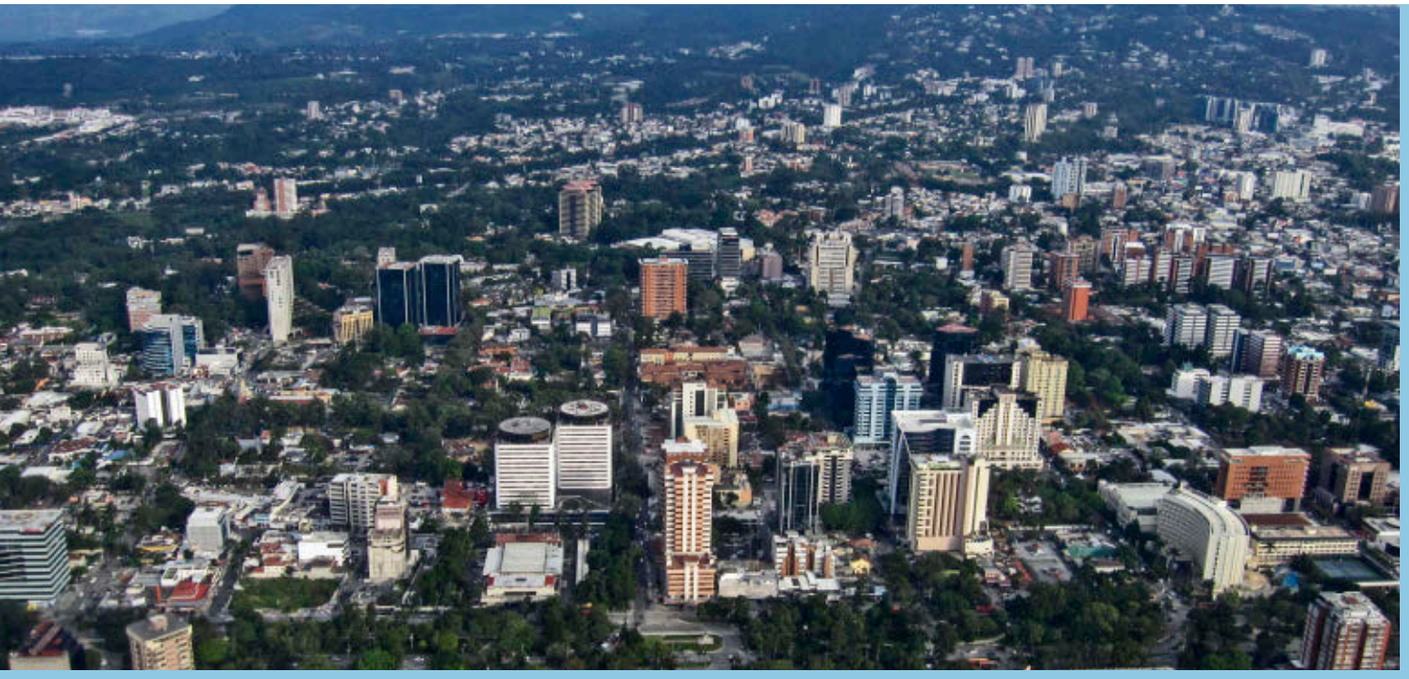
FEATURED JAVA SPECIFICATION REQUEST

JSR 364: BROADENING JCP MEMBERSHIP

JSR 364: Broadening JCP Membership is the fourth component of the JCP.next effort that was initiated by the Java Community Process (JCP) team in May 2011. JCP.next seeks to increase transparency, participation, agility, and governance within the JCP. The specific goal of JSR 364 is "to broaden JCP participation by defining new membership classes, changing existing membership categories, enabling participation by the community, and ensuring the appropriate Intellectual Property commitments from JCP Members."

Heather VanCura (pictured) is the Specification Lead for JSR 364. The Expert Group includes representatives from the London Java Community (LJC) and SouJava, along with representatives from small and large companies, all of whom are seeking greater engagement and interaction between the JCP and the broader Java developer community.

If you'd like to follow the progress, comment on the work, or contribute new suggestions, visit the JSR 364 project on Java.net. There, you'll find a variety of means for bringing your own ideas into the effort.



FEATURED JAVA USER GROUP

GUATEMALA JAVA USERS GROUP

The Guatemala Java Users Group (GuateJUG) held its first meeting on March 3, 2011. Initially, the JUG was led by **Maria Castillo** (@marycoder), **Mario Batres** (@mariobatres7), and **Víctor Orozco** (@tuxtor). "Since then," Orozco notes, "the core team has only grown as the JUG grows!"

Today, GuateJUG has more than 580 members, including Java developers, architects, and other enthusiasts and users. About 150 members actively participate by attending meetings as well as conversing on the mailing list.

The group meets on a monthly basis. A typical meeting includes a technical talk, along with informal socializing, to provide networking opportunities and also to grow the group's membership. The technical talks are often centered on Java tools and trends, with variety added by guest speakers who make presentations based on their own specific areas of expertise.

GuateJUG is also active beyond its regular monthly meetings. For example, the group participates in Java launch events; gives presentations at universities, technology incubators, and other user groups; and organizes social trips ("Yes, we also like to hike volcanos!" Orozco declares).

One of GuateJUG's major accomplishments occurs on an annual basis: Java Day Guatemala has become the biggest Java conference in Central America. More than 500 developers attended the 2014 conference in October. Attendance has been increasing every year.

"GuateJUG is recognized as one of the strongest developer groups in Guatemala," says Orozco. "Our country is gaining attention as a technology hub, since it offers many facilities for outsourcing and software development. Java usage by government and educational organizations is increasing rapidly. So, within this context, Guatemala is becoming a great place to be a Duke advocate."

Learn more about GuateJUG on Java.net, Facebook, and Twitter.

PHOTOGRAPH BY VISUALISTICA/FLICKR

JCP Election Results

Results are in from the 2014 Fall Executive Committee Elections for the Java Community Process (JCP). The election was hosted by Votenet and closed on November 10, 2014, at 11:59 p.m. Pacific Time.

This year, there were eight ratified and five elected seats open for election. Because the JCP is transitioning to JCP 2.10 in 2015, all those who won elected seats will serve a one-year term.

The following JCP members were elected or re-elected to ratified seats: Freescale, Gemalto M2M GmbH, Goldman Sachs, MicroDoc, SAP, Software AG, TOTVS, and V2COM.

These members were elected or re-elected to open election seats: ARM; Azul Systems; Hazelcast; Werner Keil; and Geir Magnusson, Jr.

Newly elected members took their seats on November 25, 2014.

Get detailed election results.

JAVA CHAMPION PROFILE

DAN ALLEN

***Dan Allen** is an author, speaker, and innovator. He participates in the Java Community Process (JCP), and has contributed to open source projects including Arquillian and JBoss Forge. He became a Java Champion in March 2013.*

Java Magazine: Where did you grow up?

Allen: In the Maryland [United States] suburbs.

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

Allen: Playing with Logo (aka turtle graphics) in kindergarten. My obsession with hacking began much later when I discovered programming on the TI-82. I started hacking on a blackjack program someone else started, and then created a few games of my own, including poker and horse racing. It was also my first taste of open source.

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

Allen: Unofficially, TI-BASIC on the TI-82. Officially, PHP, which I learned from reading books in a nearby bookstore during graduate school.

Java Magazine: What was your first professional programming job?

Allen: I left a PhD program at University of California, Santa Barbara, in 2001 to work for a startup company creating web-based benefits software. We eventually converted the program to PHP. During that time, we discovered Ajax before it had a name.

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

Allen: Working on open source software and interacting with the community. I want to spend more time in the Colorado Rockies, but I haven't gotten around to it yet.

Java Magazine: What happens on your typical day off work?

Allen: It took me a long time to discover it, but I'm an introvert. While I love interacting with the community, whom I consider family, I also enjoy taking lunch breaks in solitude, reading, and watching shows on my phone.

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

Allen: Meeting people from all over the world and drinking beer with them. They inspire me.

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

Allen: In a subtle way, yes. I constantly think about ways to improve the platform, particularly through networking. You could say that by becoming a Java Champion, I now think about being one.

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

Allen: Helping to make open source, and those who participate in it, wildly successful.

You can find Dan Allen on Twitter as @mojavelinux, at GitHub, and on Google+.

PHOTOGRAPH COURTESY OF PHANTASIALAND

EVENTS

JavaLand *MARCH 24–26*
BRÜHL, GERMANY
This two-day conference is a gathering of Java enthusiasts, developers, architects, strategists, and project administrators. Session topics include cloud and big data, core Java, enterprise Java, front end, IDEs and tools, Internet of Things, and languages for the Java Virtual Machine. Plus, attendees get exclusive use of Phantasialand and all of its rides and attractions.

Apache Hadoop Innovation Summit
FEBRUARY 12–13
SAN DIEGO, CALIFORNIA
This two-day event offers scientists, engineers, and architects a deep dive in Hadoop innovation with keynotes, hands-on labs, workshops, and networking opportunities.

Embedded World
FEBRUARY 24–26
NUREMBERG, GERMANY
The Embedded World Exhibition and Conference brings together developers, designers, business partners, and scientists. The event focuses on innovations in embedded systems and development and covers hardware, software, and tools. More than 900 exhibitors, 22,000 visitors, and 1,500 participants are expected.

Mobile World Congress
MARCH 2–5
BARCELONA, SPAIN
This industry-leading event focuses on in-depth analysis of present and future trends in the mobile industry.

QCon London
MARCH 2–3, TUTORIALS
MARCH 4–6, CONFERENCE
LONDON, ENGLAND
QCon London is the eighth annual London enterprise software development conference designed for developers, team leads, architects, and project managers.Topics covered include Java, HTML5, mobile, agile, and architecture communities.

EclipseCon 2015
MARCH 9–12
SAN FRANCISCO, CALIFORNIA
This conference brings together the Eclipse community to share best practices and innovation in software development with Eclipse IDE.

DevNexus
MARCH 10–12
ATLANTA, GEORGIA
Organized by the Atlanta Java User Group for Java professionals, this event offers seven tracks and more than 50 sessions.

JAVA BOOKS

IRON-CLAD JAVA: BUILDING SECURE WEB APPLICATIONS

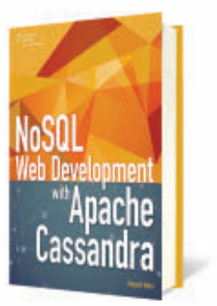
By Jim Manico and August Detlefsen
Oracle Press, September 2014

Iron-Clad Java describes the use of several Open Web Application Security Project (OWASP), Oracle, Apache, and Google open source Java projects that are essential tools needed to construct a secure web application with the Java programming language. The authors present best practices for authentication and access control, defense against cross-site scripting and cross-site request forgery, cryptographic storage, and injection protection. Using the practical advice, best practices, and real-world examples provided in this authoritative resource, readers will gain software engineering techniques for increasing security.

JAVAFX RICH CLIENT PROGRAMMING ON THE NETBEANS PLATFORM

By Gail and Paul Anderson
Addison-Wesley Professional, September 2014

Focusing on JavaFX as the front end for rich client applications, this guide's examples cover JavaFX 8 with the NetBeans Platform, NetBeans IDE, and Java SE 8. Gail and Paul Anderson fully explain JavaFX and its relationship with the NetBeans Platform architecture, and systematically show Java developers how to use them together effectively. Each concept and technique is supported by clearly written code examples, proven through extensive classroom teaching.



NOSQL WEB DEVELOPMENT WITH APACHE CASSANDRA

By Deepak Vohra
Cengage Learning, January 2015

Apache Cassandra is the most commonly used NoSQL database written in Java and is renowned in the industry as the only NoSQL solution that can accommodate the complex requirements of today's modern line-of-business applications. Cassandra is the technology of choice for such data-driven organizations as Netflix, eBay, Constant Contact, Comcast, and scores of others. In *NoSQL Web Development with Apache Cassandra*, you will learn about all aspects of using Cassandra in web applications—including accessing the Cassandra database using the common programming/scripting languages Java, PHP, Ruby, and JavaScript. Master web development using Apache Cassandra with the help of this book.

JAVA IN A NUTSHELL, 6TH EDITION

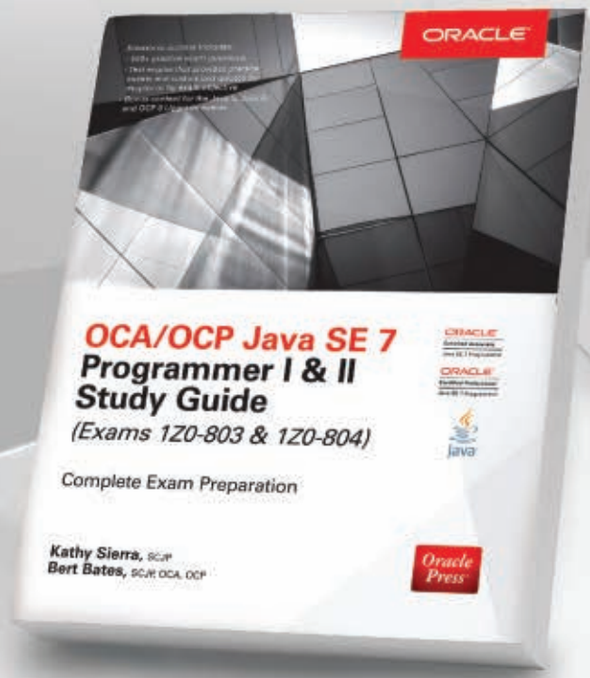
By Benjamin J. Evans and David Flanagan
O'Reilly Media, October 2014

The latest edition of *Java in a Nutshell* is designed to help experienced Java programmers get the most out of Java SE 7 and Java SE 8, but it's also a learning path for new developers. Chock-full of examples that demonstrate how to take complete advantage of modern Java APIs and development best practices, the first section of this thoroughly updated book provides a fast-paced, no-fluff introduction to the Java programming language and the core runtime aspects of the Java platform.

Oracle Press™

Your Destination for Java Expertise

Written by leading Java experts, Oracle Press books offer the most definitive, complete, and up-to-date coverage of Java available.



OCA/OCP Java SE 7 Programmer I & II Study Guide

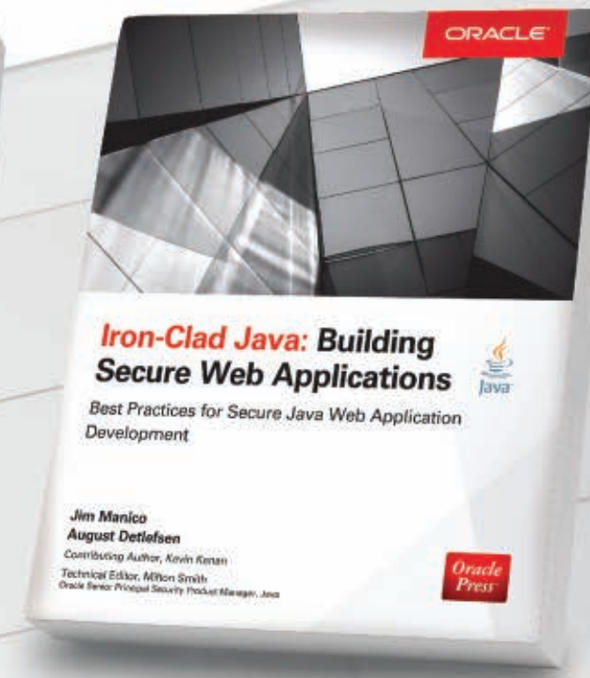
Kathy Sierra, Bert Bates

This self-study tool offers full coverage of OCA/OCP Exams 1Z0-803 and 1Z0-804. Electronic content includes 500 practice exam questions.

Java EE Applications on Oracle Java Cloud

Harshad Oak

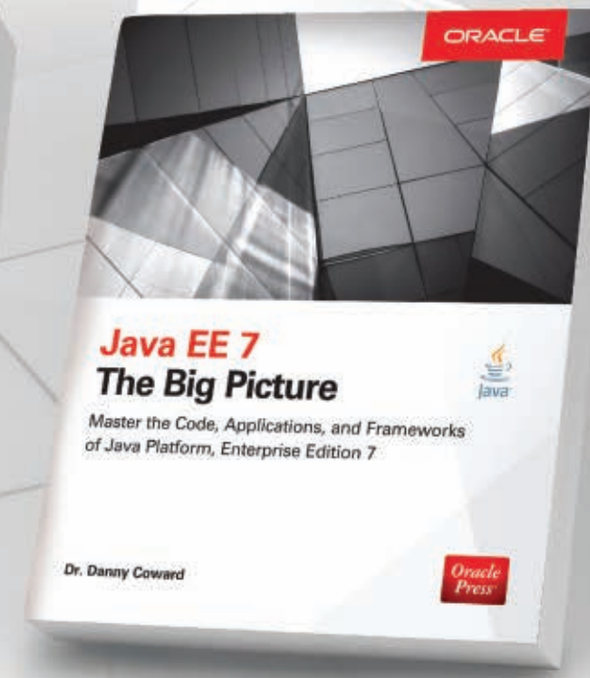
Build highly available, scalable, secure, distributed applications on Oracle Java Cloud.



Iron-Clad Java: Building Secure Web Applications

Jim Manico, August Detlefsen

Develop, deploy, and maintain secure Java applications using the expert techniques and open source libraries in this Oracle Press guide.



Java EE 7: The Big Picture

Danny Coward

Master the code, applications, and frameworks that power Java Platform, Enterprise Edition 7.

Oracle Press™ 20 YEARS

Available in print and as eBooks

Visit us at OraclePressBooks.com • Follow us @OraclePress and Like us OraclePress

From left to right: Bot-So's Debraj Dutta, Avinaba Majumder, and Tapas Bose take a break in an outdoor cafe during JavaOne.

IoT Developer Challenge Winner

Bot-So

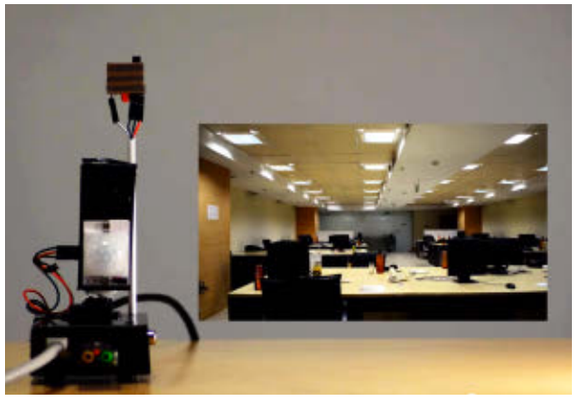
Monitor your home while you're away. **BY KEVIN FARNHAM**

Bot-So is an Internet of Things (IoT) robot that applies a Raspberry Pi, Oracle Java SE Embedded 8, a motion sensor, a video camera, and Twitter to let you monitor your home or office while you're away. A 2014 IoT Developer Challenge winner, Bot-So can be commanded to scan a room and send you a video, or can operate in surveillance mode, where it will record pictures and video when the motion sensor detects activity.

PHOTOGRAPH BY BOB ADLER/GETTY IMAGES

Bot-So was developed in India by Debraj Dutta, Tapas Bose, Avinaba Bandhu Majumder, and Suddhantha Krishnan. "We have been tinkering with IoT on open hardware for some time," says Majumder. "The main objective of this project was to realize an IoT prototype that provides ease of interaction between IoT devices and human beings, using social media."

To interact with Twitter, Bot-So uses the Twitter4J library. Other important software components include Jetty, Log4j, the Java Mail and Google Drive APIs, and the Raspbian operating system based on Debian Linux. It took the Bot-So team one and a half months to complete the project.



Watch the Bot-So project in action.

The Bot-So team attended JavaOne 2014 and showcased Bot-So on center stage at Oracle Technology Network's Tech Fest. "It was an amazing experience," says Majumder.

He believes that IoT is already showing big promise and is going to be the next big thing in IT. "As hardware and sensors become cheaper, there will be more devices connected to the internet. This will create a need for larger sophisticated systems to control the devices as well as collect and process data from them," he says. "We developed Bot-So with the idea that IoT devices could use social media to interact with human beings, since humans are already so familiar with social media. We hope to build more useful solutions based on the Bot-So reference platform, to meet both personal and industrial needs."

Give it a try. Download the Bot-So software on GitHub. `</article>`

FAST FACTS

- Bot-So is a robot that communicates using Twitter, letting you receive video updates from your home or office when you're away.
- Bot-So can operate in surveillance mode, taking pictures and videos when its motion sensor detects movement.
- The technology behind Bot-So includes a Raspberry Pi, Oracle Java SE Embedded 8, Jetty, and Twitter4J.


Your hassle-free Continuous Integration and Deployment server
Sets up in minutes—works 24/7
TeamCity
Build your success
Developed by JetBRAINS, makers of the legendary IntelliJ IDEA

At the Kenyan waterworks, a man loads donkeys with water for home delivery.

Cloud Computing

MAKING WATER SAFE

An all-Java mobile app documents Safe Water Kenya's efforts to provide clean water to rural families in remote East Africa. **BY PHILIP J. GILL**

PHOTOGRAPH COURTESY OF SAFE WATER KENYA

Clean, safe drinking water is something that people in the developed world take for granted every time they turn on a faucet. But for more than 1 billion people living in the developing world, the lack of clean, safe drinking water poses a major health threat and a significant barrier to education and economic development.

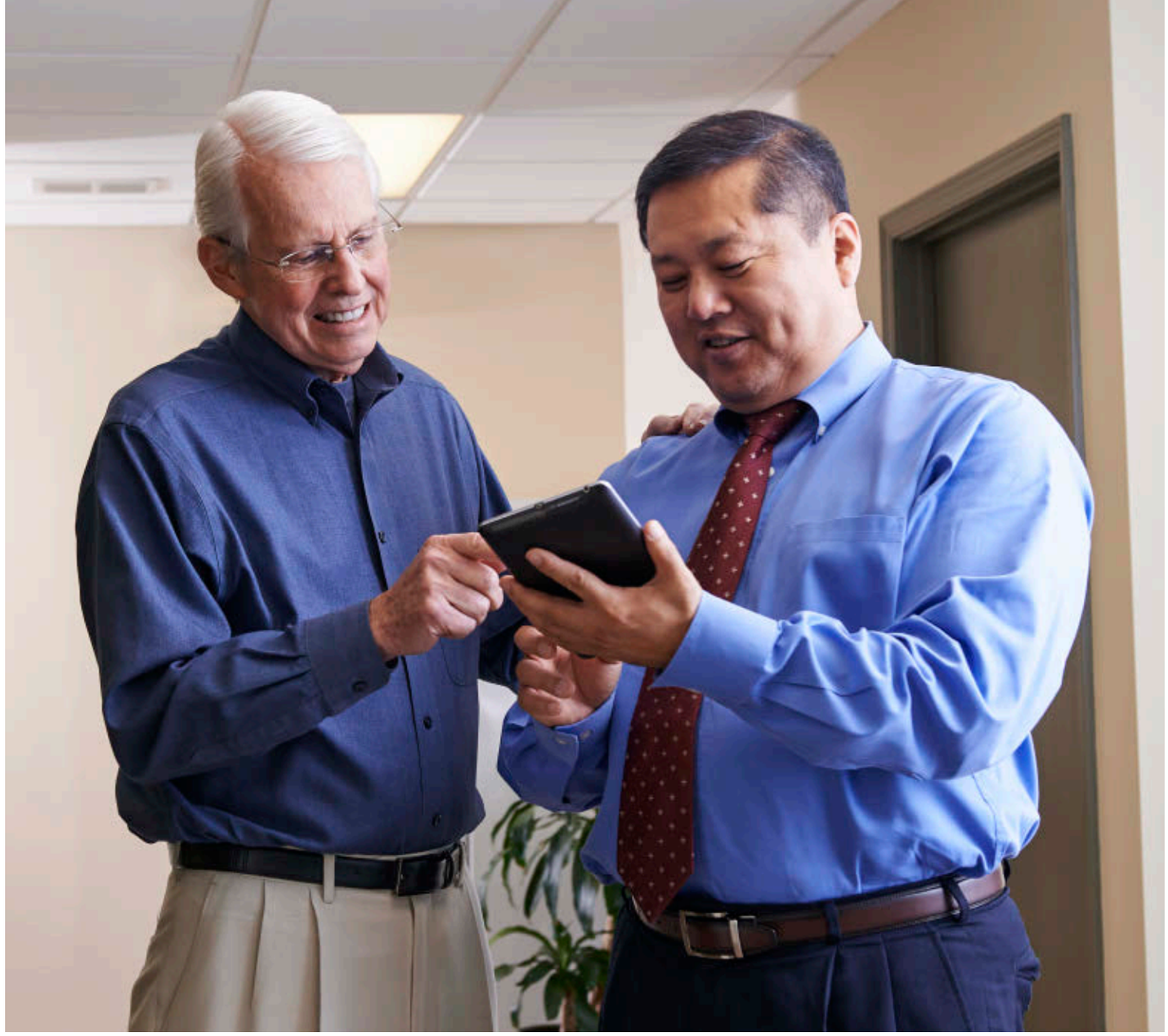
The World Health Organization (WHO) says that more than 2 million people—95 percent of whom are children—die from the consequences of a variety of waterborne illnesses each year. Indeed, WHO, a United Nations agency based in Geneva, Switzerland, reported outbreaks of cholera, the age-old waterborne scourge that causes debilitating and potentially fatal diarrhea, in more than 50 countries around the world last year.

SAFE WATER KENYA
(a project of the Safe Water Team)
safewaterteam.org

Headquarters:
Grand Rapids, Michigan

Industry:
Nongovernmental organization (NGO)

Oracle technologies used:
JDK, Oracle Database 12*c*, Oracle WebLogic Server 12*c*, Oracle Database Mobile Server, Oracle Berkeley DB Transactional Data Store



Safe Water Kenya's Don Arnold (left) takes a look at the latest Survey App, developed by mFrontiers' Daniel Pahng (right).

BIG PROBLEM
More than 1 billion people lack access to clean, safe drinking water, according to the Water Project.

PHOTOGRAPH BY ANDREA MANDEL

"It's not just a matter of illness; there's an economic factor as well," explains Don Arnold, executive director and founder of Safe Water Kenya (SWK), a project of the Safe Water Team (SWT). A nonprofit based in Grand Rapids, Michigan, SWT works to improve water quality in the developing world.

"There are not too many people in Africa with salaries," says Arnold, a veteran plumbing industry consultant with more than 30 plumbing patents to his name. "So if they get sick, they don't work, and they don't get paid for that day. If the children are sick, mommy has to stay home to care for them, and she doesn't get paid that day, and the children don't go to school."

To address these critical water issues in rural Kenya, an East African republic, SWK began installing Hydraid BioSand Water Filters—based on the low-tech, slow sand filtration process—more than a year ago. "We have installed 2,500 [filtration systems] so far," says Arnold. "We figure the average family has seven people, so that in just a matter of a year or so, we have affected 17,000 lives."

Those filters don't come for free. "To document the installations to our donors, we have to fill out an extensive survey that includes photos, GPS coordinates, and a signature from the

Arnold (in red shirt) with Safe Water Kenya warehouse workers and installation team members

PHOTOGRAPH COURTESY OF SAFE WATER KENYA

receiving party," says Arnold.

SWK meets these requirements using a decidedly high-tech solution: an all-Java "Survey App" that runs on handheld Android tablets developed specifically for SWK by mFrontiers, a Libertyville, Illinois–based enterprise software house and Oracle partner. The app earned mFrontiers a 2014 Oracle Excellence Award for Sustainability Innovations.

CARBON FOR CLEAN WATER

Arnold was inspired to start SWK while on a church mission to Africa six years ago. "I was in Uganda riding in a car with a local leader, and he stopped alongside the road and pointed down to a river where people were filling plastic containers with filthy brown water," Arnold recalls. "He told me that the people would take that water home, drink it and cook with it, and get sick. In a lot of cases, he said, they really didn't understand where waterborne diseases came from."

To get started, SWK shipped 2,250 Hydraid filters to Kenya, expecting them to last 18 months. But thanks to an energetic ground operation led by SWK local country manager Vivian Akinyi, who works through rural village health clinics to find needy families, SWK passed that mark in nine months and has not looked back since. "Today we install 15 filters on a typical day," says Arnold.

Initially, major funding for the water filters and installations came from carbon credits certified by the non-profit Gold Standard Foundation, also based in Geneva. "Our current credits are granted on the basis that we use a device or technology that reduces carbon emissions," explains Arnold. "In our case, we are replacing burning wood or charcoal to boil water to make it safe to drink. Every time we put a filter in a home where they were boiling water, we're reducing emissions by that much."

Originally, the SWK surveys were completed manually, using pen, paper, camera, and a GPS device in the field, and the data was then re-entered into a computer system back in SWK's distribution office for transmission back to SWK headquarters in the US. But that process was time-consuming and potentially error-prone. Today, the SWK Survey App collects and stores data on an Android tablet running in offline mode, automating the process and reducing errors. mFrontiers developed the Survey App, written entirely in Java and deployed on an all-Oracle product stack, based on its mFinity Enterprise Mobility Management Platform.

Low-Tech Clean Water

The Hydraid BioSand Water Filter that Safe Water Kenya (SWK) uses is a low-tech, sustainable device designed to provide clean, safe water for drinking, cooking, and sanitation. Created by David Manz, a professor at the University of Calgary, in Alberta, Canada, the Hydraid filter builds on the slow sand water filtration process originally developed in Scotland in the early 1800s and used to provide the first public clean-water system in London in 1829.

Slow sand filters use naturally occurring biological processes to clean water; a gelatinous layer or biofilm that grows naturally on the surface of a few millimeters of sand eliminates contaminants from any water source, from lakes and streams to rainwater. "The good bacteria eats the bad bacteria," explains SWK Executive Director and Founder Don Arnold.

"There are a lot of technologies that will work to provide clean water, but few are as practical as the Hydraid filter because it doesn't require replacing elements such as cartridges or adding chemicals such as chlorine," Arnold adds. "The Hydraid filter eliminates 95 percent of bacteria, viruses, and parasites in the water. Even chlorine can't remove parasites."

Manz' second innovation in slow sand filters was to replace the large, heavy concrete containers traditionally used with a lightweight, easy-to-install, FDA-approved plastic container. That container measures 30.5 inches tall and 16.5 inches in diameter, and weighs 8 pounds empty and 140 pounds when filled. Installation takes 30 minutes, and the filter produces up to 47 liters per hour, enough for a family of 8 to 10 people. The biological layer is triggered when the first water is poured on top, and it takes about two weeks to fully form. It uses no electricity and provides clean, safe water for 10 years with little maintenance (simply removing the solids that collect at the top from time to time).

PHOTOGRAPH COURTESY OF SAFE WATER KENYA

Arnold and Safe Water Kenya local country manager Vivian Akinyi (left) visit a cooperating health clinic.

After installing the water filter, the SWK worker fills out the survey. "The survey consists of seven or eight pages on an Android tablet, each of which has five or six questions," says mFrontiers President Daniel Pahng, who wrote the Survey App himself using jQuery, an open source JavaScript library for client-side apps. "Using the tablet, they [SWK workers] also take photos of the family to add GPS coordinates because there are no street addresses."

As a final step, the installer records the recipient's signature and the serial number of the Hydraid water filter on the tablet form and moves on to the next home.

Because there are no internet connections in these remote locations, the data is stored on the Android tablet in a light-footprint Oracle Berkeley DB datastore. "When they return to the distribution office," says Pahng, "the Oracle Database Mobile Server sync agent, running on the Android tablet, automatically uploads survey information to an Oracle database hosted on a cloud-based Oracle WebLogic

Left: A Hydraid BioSand Water Filter is installed using sand and gravel. Right: A Safe Water Kenya graduation ceremony for trained health workers.

PHOTOGRAPHS COURTESY OF SAFE WATER KENYA

Server, from which the information may be accessed using a browser back at Safe Water Kenya's headquarters."

Pahng notes that the mFinity platform was originally written in Microsoft .NET. "However, when the company decided to expand globally, we realized that Microsoft was not a leader in the mobile world," he explains. "For most enterprise-level customers, we needed Java."

UNPRECEDENTED TRANSPARENCY

Not only does the Survey App speed the installation process; it also creates a level of transparency that few nonprofits can match, says SWK's Arnold. "Many nonprofit organizations really don't give their donors more than general information," he explains. "In other words, if they were installing filters like we are, maybe in a once-in-a-while newsletter or maybe the annual report they could say they were pleased to report that 'last year we installed X number of filters.'

"Well, that's good, but it isn't as good as what we can do," Arnold continues. "The very night that the filters are installed, the survey is uploaded to our system and people all over the world, our staff and our donors, can see what we did. They see pictures of the families and all sorts of information about them. I don't know any other organization that has a tool like that."

In addition to quickly informing staff and donors, Arnold expects the app and tablet computers to help SWK accelerate its work. "We're trying to be practical about what we can take on," says Arnold, "but we're planning on doubling this year to about 5,000 filters."

"We're very grateful to mFrontiers and Oracle," adds Arnold. "The system is beyond anything that we were smart enough to ask for." `</article>`

MORE ON TOPIC:

Cloud Computing

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

BRUNO SOUZA AND
EDSON YANAGA

BIO

Three Steps to Improve Your Career

Code more effectively, build communities, and explore the cloud to boost your skills and network.

We all know that success takes time and effort. In *Outliers*, Malcolm Gladwell made famous the "10 thousand hours" rule—that is, we need to invest roughly that amount of time to succeed in any chosen endeavor (at least to succeed in a big way, if the examples of Bill Joy and Steve Jobs are anything to go by). But 10,000 hours is an overwhelming number. All we want is to improve our career day by day.

We don't need that much time to get the ball rolling. Success is a combination of many small, concentrated efforts. In *The First 20 Hours*, Josh Kaufman suggests that you can get amazing results by putting 20 focused hours of effort into something. And 20 hours isn't so bad. We can do that.

As developers, just improving in a few broad areas—code, community, and cloud computing—can push our career forward. If you can invest 20 hours in these three areas, your career can get the boost you're looking for. Try our exercises at home. The results, we think, will amaze you.

CODE: EMPLOY CODE KATAS

What do music, sports, martial arts, and programming have in common? Greatness is achieved through practice. It helps to learn the theory, techniques, and tools. But to gain mastery, we need practice and feedback.

If you want to be a great developer, you need to code. A lot. Sounds obvious, right? When we say that you need to code more to become a better developer, it sounds simple enough. Coding is mostly what we do. But the reality is that most of the code we write every day doesn't challenge us. If we don't push ourselves, we won't improve. Writing large amounts of simple code that doesn't promote different thinking isn't helpful.

In martial arts, a *kata* is a teaching method. By constant repetition, students learn to do the movements in a natural way. In programming, Dave Thomas coined the term *code kata*. It amounts to solving a series of small problems, expanding and refining your code in every iteration. The purpose is not to achieve a perfect solution, but to sharpen programming skills through practice.

PATH TO OPTIONS
"With repetition, **the alternate approaches become clear,** options open."
—*Robert Genn*

ShuHaRi, a philosophy with its roots in the martial arts, has been popularized in the agile development world. The basic idea is that knowledge comes through three

Edson Yanaga introduces the concept of code katas and shows you an example.

Cloud Computing

BRUNO SOUZA PHOTOGRAPH BY BOB ADLER/GETTY IMAGES

sequential stages:

- *Shu*: You blindly follow the kata without worrying about the why.
- *Ha*: After you know the practice, you can learn the theory and techniques behind it. You learn from other sources and integrate them into your practice.
- *Ri*: You've mastered your craft and now can innovate, creating your own approaches.

Athletes and musicians need intensive training before a match or a concert. But in the programming world, developers commonly go to battle without preparation.

JAVA 8 STREAMS AND LAMBDAS KATA

No doubt you want to practice right now with some new Java 8 code. To help you get started sharpening your skills, we prepared a simple Java 8 Streams and Lambdas Kata (JUnit tests included). You can download the code kata source code on GitHub.

If you're not familiar with lambdas, we encourage you to pass the tests using the imperative syntax first. Then you can refactor your code with small steps to lambdas and ensure that your tests are still passing. Enjoy!

Code katas are practice time for developers.

Code katas work well with test-driven development and test frameworks such as JUnit and TestNG because they can help you with an iterative and progressive way of solving problems.

Right now, many developers are trying to learn the new features of Java 8. This is the perfect opportunity to practice some code katas. So, take a first step to improve your skills.

ACTION ITEM: Practice the Java 8 Streams and Lambdas Kata (see **sidebar**).

COMMUNITY: JOIN A JAVA USER GROUP

Don't stagnate! Get involved with the larger developer community. Sharing and collaborating with other developers are great ways to push us outside our comfort zone, increasing the pace and breadth of our learning.

The Java community—built on the notion of openness, diversity, and inclusion—is fascinating. It encompasses multiple platforms and many vendors, languages, and people from all over the world. The Java community is a powerful place to explore and grow as developers.

We need to study and improve our skills, but if we aren't challenged, if we don't get exposure to new ideas, if we aren't questioned about our choices, we vegetate. Joining a cohesive collection of people with the same interests and passions who ask the hard questions is a strong driver to improve our career and our personal life. Joining an active Java user group (JUG) is an easy way to find friends who have similar problems and different solutions, or similar solutions to different problems. Exchanging experiences and opportunities changes our perceptions and challenges us to try something different.

DON'T FLY SOLO

"If you want to go quickly, go alone. **If you want to go far, go together."**

—African proverb

Networking within developer communities is a challenging and fun experience. Practical activities such as hackathons, Adopt-a-JSR, Dojos, and lightning talks promote bidirectional learning. Those JUG activities can raise everyone's abilities in coding, thinking, and presenting.

ACTION ITEM: Join a JUG mailing list and attend the next meeting.

ACTION ITEM: Join a practical, hands-on activity in your local JUG. If it doesn't have one, propose one and help organize it. Even small study groups can make a huge impact on your career.

THINK DIFFERENTLY: EXPLORE CLOUD COMPUTING

Great developers are open-minded. The more ideas our brain gets exposed to, the more relationships it can make, which helps us solve large problems. Some technologies can expand our mind to new horizons, even if we don't work with them at our daily jobs.

Programming is the art of using your skills to craft solutions in a restricted environment. In each of these environments, you choose between a finite set of tools and techniques, which shape your view of the solution. As time passes, the world, and the restrictions, change. New restrictions suggest new tools and techniques to craft the solution, and the ball keeps rolling.

Yes, there is a lot of talk about cloud computing and you might be tired of hearing about it. But don't be put off by that: the concepts around the cloud create new ways of thinking and can expand our abilities and create new opportunities for our career.

As developers, we deal with computers, handle memory addresses, and access underlying storage systems. Take away the computer. Remove the memory. Get rid of the

storage. Turn everything into networks. What happens then to our applications? Cloud computing, big data, microservices, containers, platform as a service (PaaS). New approaches that deal with computers that don't exist, memory and storage that aren't there or pretend to be infinite.

Of course, we know that underlying it all there are real computers somewhere, and application servers and operating systems. But the goal here is to take a step back and experiment with a new way of thinking.

The cloud is an opportunity to think differently. Pretend that there are no limits. Now things that seemed impossible to create become possible. We can unleash our creativity.

The promise of the cloud allows us to think *productively*, not *reproductively* (as in "repeating the past"). The reality may not be so sweet, but we can pretend it is.

To open our minds to new ways of thinking, we can practice suspension of disbelief. Experiment with cloud computing, and spend time and effort to create your code in an environment where everything could be infinite. This view may not fix your immediate problems at work, but it will push your career forward in interesting ways.

ACTION ITEM: Create an account with a cloud computing provider and deploy some code.

ACTION ITEM: Experiment with different kinds of clouds, from containers to PaaS to big data and Hadoop clusters.

OPEN WIDE

"The mind that opens to a new idea never returns to its original size."

—Albert Einstein

PUT IT ALL TOGETHER

Of course, our efforts are a lot more fun when we mix it up. How about proposing a code kata session to your JUG? Presenting your experiences with cloud at the next meeting? Or proposing a group discussion of code katas with solutions in cloud environments?

Every time we think differently—when we try to see all the possible solutions to a problem—we grow as developers. Spend 20 hours in each of those areas, and see what a difference it makes. `</article>`

MORE ON TOPIC:

Cloud Computing

LEARN MORE

- code katas
- The Creativity Post
- Join a JUG

MICHAEL KÖLLING

BIO

Part 2

Code Java on the Raspberry Pi

Access Raspberry Pi hardware interactively with BlueJ.

In Part 1 of this series, we discussed how to run BlueJ on the Raspberry Pi to do some Java programming directly on the pocket-size computer, without the need for a second machine. We no longer need to develop our Java programs on a desktop machine and then transfer the JAR file to the Raspberry Pi; the promise of using the Raspberry Pi as a desktop replacement for tinkering and learning with Java is here.

However, in Part 1, we talked only about doing on the Raspberry Pi what you can also do on your desktop: run BlueJ and some standard Java examples, and use the same functionality that has been available on desktops and laptops for some time.

Now, don't get me wrong: There is nothing wrong with using BlueJ on the Raspberry Pi just to learn how to write standard Java programs. This combination is a great educational tool, and the fact that the Raspberry Pi is available so cheaply is a game changer. However, we do not need to stop there.

The next exciting step is to do some things that you could not do before: access some of the physical components of the Raspberry Pi directly from Java, and play not only with code but also with hardware. The combination of easily accessible hardware on the Raspberry Pi and the interaction and experimentation functionality of BlueJ allows us to do some cool things.

In this article, we will explore how to create Java objects that correspond directly to hardware parts and are easy to use.

What You Need

I am assuming that you have set up your Raspberry Pi, you can log in and start the X Windows server, and you have installed and started BlueJ. (If not, read Part 1 of this series first.)

In addition, to follow along with the examples in this article, you will need a few additional items: one LED, one button, one resistor, and four wires (see **Figure 1**). A breadboard also helps—it is not strictly essential, but if you don't use a breadboard, it can be difficult to attach the components to each other without soldering. If you try to just twist the ends of the wires and the components together, they tend to come apart very easily.

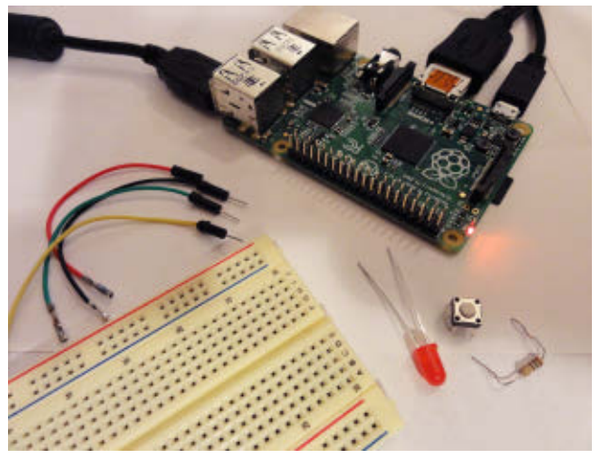


Figure 1

Getting the BlueJ Project

Let's start by opening a BlueJ project. Download the RasPi-IO project here, unzip it, and open it in BlueJ on your Raspberry Pi (see **Figure 2**).

This project has two classes that provide abstractions over I/O components that can be connected to the Raspberry Pi: a GPOutput class for anything that can be connected to a general-purpose input/output (GPIO) pin (such as an LED), and a Button class for a push button connected to a GPIO pin.

Preparing an LED

The first, very simple example we'll discuss is switching an LED on and off.

Start by connecting your

PHOTOGRAPH BY JOHN BLYTHE

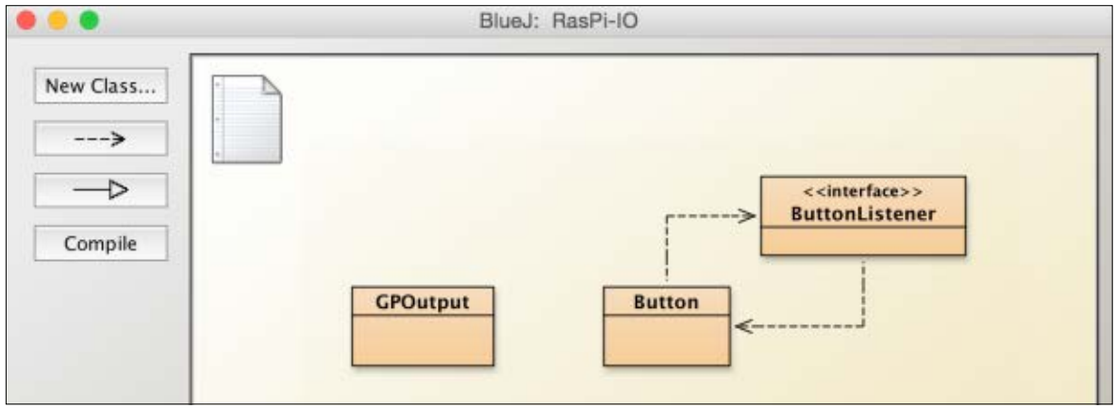


Figure 2

LED to the Raspberry Pi in the following way:

1. Connect the ground terminal of the LED (the shorter leg) to a wire.
2. Connect the end of the wire to pin 6 (Ground) of the Raspberry Pi's expansion header. Refer to the pin configuration diagram for the Raspberry Pi.
3. Connect the longer leg of the LED to the resistor.
4. Connect the other end of the resistor to a second wire.
5. Connect the other end of the second wire to GPIO 1 (pin 12) of the Raspberry Pi's expansion header.

The whole setup is shown in **Figure 3**.

Be careful that you are using the correct pin numbering scheme. Unfortunately, different numbering schemes for pins are in use. BlueJ uses the Pi4J library, so we'll use the numbering scheme used by that library. Make sure you are looking at the right one (see the diagram mentioned earlier). GPIO 1 should be pin 12.

Getting to Work in BlueJ

Now that we have connected our LED to one of the GPIO pins, in BlueJ, create an object of class GPOutput. (Remember, you can create an object in BlueJ by right-clicking the class and selecting a constructor from the menu.)

You will see two constructors: One lets you select the GPIO number you want to use, and the other uses GPIO 1 by default. Because we have connected our LED to GPIO 1, we can just use the default constructor. We now have an object on the object bench that represents our LED. (In the general case, the object represents whatever is connected to GPIO 1; for us, that is the LED.)

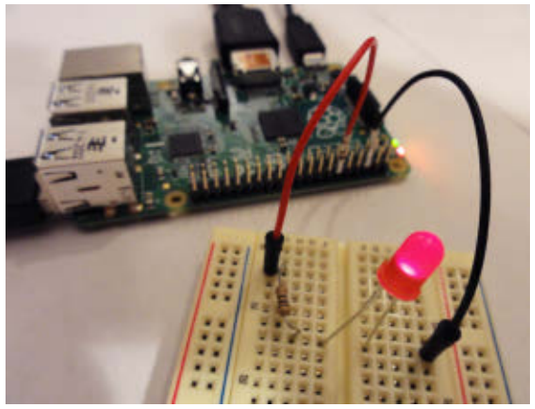


Figure 3

You can now control the LED by interacting with the object. It's that easy! Right-click the gPOutput object and invoke the on() method (see **Figure 4**). The LED should come on. Switch it off again. There is also a blink() method that you can play with. Try it with some different parameter values.

Using a Button

Next, we will connect a button to our Raspberry Pi and read its state. Connect your button as follows (see **Figure 5**):

1. Using the third wire, connect one connector of the button to 3.3V power (pin 1) on the Raspberry Pi's expansion header.
2. Using the fourth wire, connect the other connector of the button to GPIO 4 (pin 16) on the Raspberry Pi's expansion header.

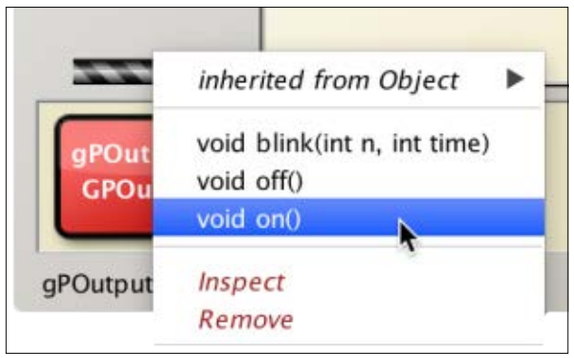


Figure 4

In BlueJ, we can represent the button by creating an object of type Button. Try it out. Again, there is a default constructor and a second constructor with a parameter that lets us specify which GPIO number we want to use. And again, we have connected our button to the GPIO that is used by default—GPIO 4, in this case—so we can make our life easy and use the default constructor.

The button object gives you two methods to check the button's state: isUp() and isDown(). If you call them without touching the button (that is, while the button is not pressed), isUp() should return true and isDown() should return false.

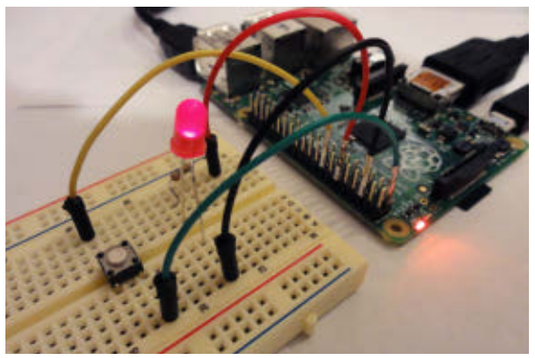


Figure 5

ACCESS TO I/O

You can use BlueJ on the Raspberry Pi not only to write standard Java programs just as you can on your desktop, but also to access the Raspberry Pi's I/O ports.

Try pressing and holding the button down while you call these methods. You should be able to see that the button's state gets correctly reported by your button object.

Writing a Program

We have seen how we can use BlueJ's interaction features to test our inputs and outputs very easily—without writing a single line of code. Next, we shall look at how to use objects of the GPOutput and Button classes in a simple program. To do this, we will write code that turns our Raspberry Pi into a light switch: The LED should come on while we press the button.

In your BlueJ project, create a new class called LightSwitch. We start by declaring two fields—one for the LED and one for the button:

```
private GPOutput led;
private Button button;
```

Then, in the constructor of your new class, create objects for each of these types, and register your own LightSwitch object as a listener for the button:

```
public LightSwitch()
{
  led = new GPOutput();
  button = new Button();
  button.addListener(this);
}
```

To make this work, you must ensure that your LightSwitch class correctly implements the ButtonListener interface by declaring this in the class header:

```
public class LightSwitch
  implements ButtonListener
```

You must also implement the buttonEvent method:

```
@Override
public void buttonEvent(
Button button)
{
  // code to come here
}
```

This listener method will be called whenever the button's state changes—both when the button is pressed and when it is released. We can then use the button parameter to check the button's state.

Now, all that remains to be done

LISTING 1

```
public class LightSwitch implements ButtonListener
{
  private GPOutput led;
  private Button button;

  /**
   * Create a Raspberry Pi light switch.
   */
  public LightSwitch()
  {
    led = new GPOutput();
    button = new Button();
    button.addListener(this);
  }

  /**
   * Listener method: called when button state changes.
   */
  public void buttonEvent(Button button)
  {
    if (button.isDown()) {
      led.on();
    }
    else {
      led.off();
    }
  }
}
```

Download all listings in this issue as text

is to react correctly when the button is pressed by switching our light on and off. We do that by entering the following code in the body of our buttonEvent method:

```
if (button.isDown()) {
  led.on();
}
else {
  led.off();
}
```

Try it out: Complete your class, compile, and create an object of type LightSwitch. (If you have trouble, compare your code to **Listing 1**, which shows the complete class discussed here.) Because the object has registered itself as a listener, you do not need to invoke any methods. Just press the button, and you should see the LED light up.

It really is that easy.

Conclusion

You can use BlueJ on the Raspberry Pi not only to write standard Java programs just as you can on your desktop, but also to access the Raspberry Pi's I/O ports. In the example we explored in this article, two wrapper classes were provided for BlueJ to represent an LED connected to an output pin and a button connected to an input pin. These wrapper classes made it very easy to interact with and write code for these components.

BlueJ internally uses the Pi4J library to communicate with the Raspberry Pi. You can either use BlueJ's wrapper classes to communicate with your hardware, or you can program with the Pi4J interface directly. (Look inside the GPOutput class to see an example of accessing Pi4J.)

Using BlueJ's wrapper classes is generally simpler—they provide a simplified interface to your hardware. More wrapper classes are available on the BlueJ website. However, the simplicity is achieved at the price of flexibility. If you need more control over the hardware than the BlueJ-provided wrapper classes give you, you can access the Pi4J library directly. This way, BlueJ manages to serve two audiences: It makes getting your feet wet very easy for people just starting out, and it gives you full flexibility after you have become more comfortable with programming slightly more-complex examples.

So, whatever your level, dive in and start writing Java on the Raspberry Pi! `</article>`

LEARN MORE

- BlueJ
- BlueJ on the Raspberry Pi
- Pi4J
- Raspberry Pi

HARSHAD OAK

BIO

Get Started with Oracle Developer Cloud Service

Run your entire development environment in the cloud.

Cloud Computing

The cloud paradigms of infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS) are now well entrenched, and most cloud vendors are looking to go beyond these services and cater to the many other aspects of building and running software. It seems only a matter of time before most of the tools and technologies required for software development are available in the cloud as a development environment as a service (DEaaS).

Developers already use the cloud in some form or another. However, in most cases, the actual development environment continues to be off the cloud. So from project management to version control, testing, bug tracking, code reviews, and continuous integration, the development process is still running in-house. Furthermore, sometimes myriad products from multiple vendors come together to constitute a software development environment.

IN THE CLOUD

Except for developers' IDEs, which will continue to run on their machines, everything else is **meant to move to the cloud.**

That's precisely the situation that's being targeted by the emerging DEaaS solutions. It's not a revolutionary idea, as such, and you probably have encountered it already in some shape or form and referred to with a different buzzword. However, the difference this time is that developers and technologies are far more capable of using remote cloud-based environments, and these new offerings are looking to move the entire development environment to the cloud, not just some parts of it.

So, except for developers' integrated development environments (IDEs), which will continue to run on their machines, everything else is meant to move to the cloud. The IDE might also move there someday, but as of today, cloud-based IDEs don't seem to quite match up to the richness of desktop IDEs such NetBeans, Eclipse, or Oracle JDeveloper.

Apart from the cost savings that cloud solutions offer, it is important to note that a cloud-based development environment also frees up some people on your team: those who are either explicitly or implicitly responsible for running or managing some aspect of the development environment. So if something in the development environment fails, you no longer would shout out to your colleague; instead, you would call the cloud vendor. All the developers on your team could stay focused on building the solution and not expending their energy on running the allied services or provisioning the infrastructure.

A Solution Must Support the Full Development Lifecycle

One of the most challenging requirements for such a cloud development environment is that it needs to support the

full software development lifecycle along with supporting developer collaboration. A cloud development environment should also alleviate the need for developers to share project-related documentation using means outside the development environment.

Having said that, the very definition of what constitutes the full development lifecycle keeps changing every few years. So it is vital that a cloud-based development environment be capable of adapting as newer development techniques and technologies appear. Think of which tools were most widely used for builds, continuous integration, and source control over the past 5 to 10 years. Are they still popular, or have they been replaced?

Let's now delve into some of the specifics of Oracle's cloud-based development environment, Oracle Developer Cloud Service.

Oracle Developer Cloud Service

Oracle has made a major cloud push over the past few years. Today, most of Oracle's products—from Oracle's many SaaS products to Oracle Database and Oracle WebLogic Server—are available in the cloud, often in various customizable shapes and forms. However, Oracle Developer Cloud Service is different because it isn't an existing offering being moved to the cloud; rather, it is a new offering built to cater to the needs of the modern development environment.

The big benefit of being a new service is that there are no issues with backward compatibility and no need to support outdated technologies that only a tiny segment of developers might need. So unlike cloud services that have evolved from existing products, Oracle Developer Cloud Service has a clean, minimalistic feel to it. It has minimal clutter and supports only what's expected of a development environment in 2015.

The following are the key features of Oracle Developer Cloud Service:

- Source control using GIT and integration with GitHub repositories
- Support for the Maven repository
- Task and issue management, with the ability to link bugs and tasks to your code
- Continuous build integration using Hudson
- Ability to deploy directly to Oracle Java Cloud Service or on premises
- IDE integration with Eclipse, NetBeans, and Oracle JDeveloper
- User and team management capability
- Wiki-based collaboration capability
- Code reviews with collaboration and filtering capability

Oracle Developer Cloud Service currently offers a few templates for common tasks and flows, but because a cloud development environment could analyze usage across hundreds of projects, I would expect Oracle Developer Cloud Service to progress to providing templates and best practices for many routines in a software development lifecycle.

No part of Oracle Developer Cloud Service is revolutionary in itself, but the fact that you can now run your entire development environment in the cloud is revolutionary. In addition, when you create a new project, the entire Oracle Developer Cloud Service stack is provisioned for you in a fraction of the time as compared to what it would take if you were creating the development environment yourself.

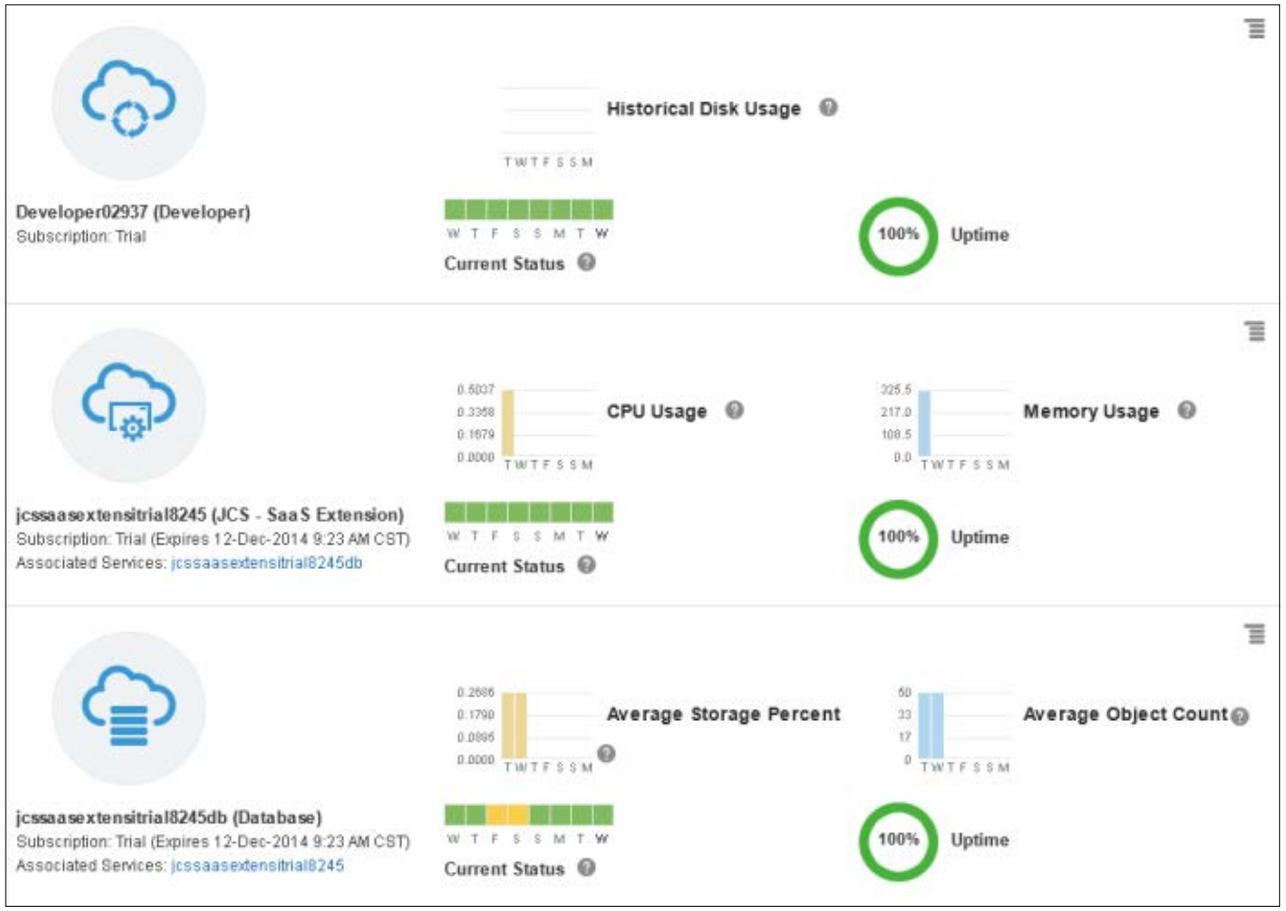


Figure 1

Trial Access

Oracle Developer Cloud Service is currently included free of cost with the trial version and paid orders of Oracle Java Cloud Service - SaaS Extension and with paid orders of Oracle Messaging Cloud Service.

The Oracle Developer Cloud Service trial includes three developer service projects and one concurrent Hudson build. For storage, it provides 1 GB for Git and Maven, 1 GB for Hudson, and 1 GB for tasks and for documentation collaboration through the wiki.

The Oracle Developer Cloud Service that's included with the paid Oracle Java Cloud Service - SaaS Extension and Oracle Messaging Cloud Service includes 12 developer service projects and three concurrent Hudson builds. For storage, it provides 6 GB for Git and Maven, 10 GB for Hudson, and 4 GB for tasks and the wiki.

To get trial access to Oracle Developer Cloud Service, request a trial for Oracle Java Cloud Service - SaaS Extension. After your Oracle Java Cloud Service - SaaS Extension trial request is approved, you will have access to an instance of Oracle Java Cloud Service - SaaS Extension as well as access to a trial version of Oracle Database Cloud Service and Oracle Developer Cloud Service.

Trying Out Oracle Developer Cloud Service

When you have access to the trial version, log in to Oracle Cloud at http://cloud.oracle.com. Use the My Services login form, and you will be greeted with the screen shown in **Figure 1**, which is a dashboard that lists your services, current status, and usage information.

Click the link to Oracle Developer Cloud Service from the services listed and you will get more information about usage and metrics, as shown in **Figure 2**. Click **Open Service Console** and then, on the console page, click **Create Project** and use the simple form that pops up to create a new project named *Java Magazine*.

After the project is created, you get a menu (see **Figure 3**) that will take you to the various services available to your project. The dashboard provides a quick project-wide status; the Tasks section lets you manage your project tasks; the Browse section is for your Git repository; the Build section is where you can create and run Hudson builds; the Reviews section is for code reviews; the Deploy section is where you can add deployment configurations, including deployment on Oracle Java Cloud or on premises; and the Wiki section is for collaboration. There are also sections for team management and administration.

NetBeans Integration

Although the Oracle Developer Cloud Service web interface is easy to use, many developers prefer to

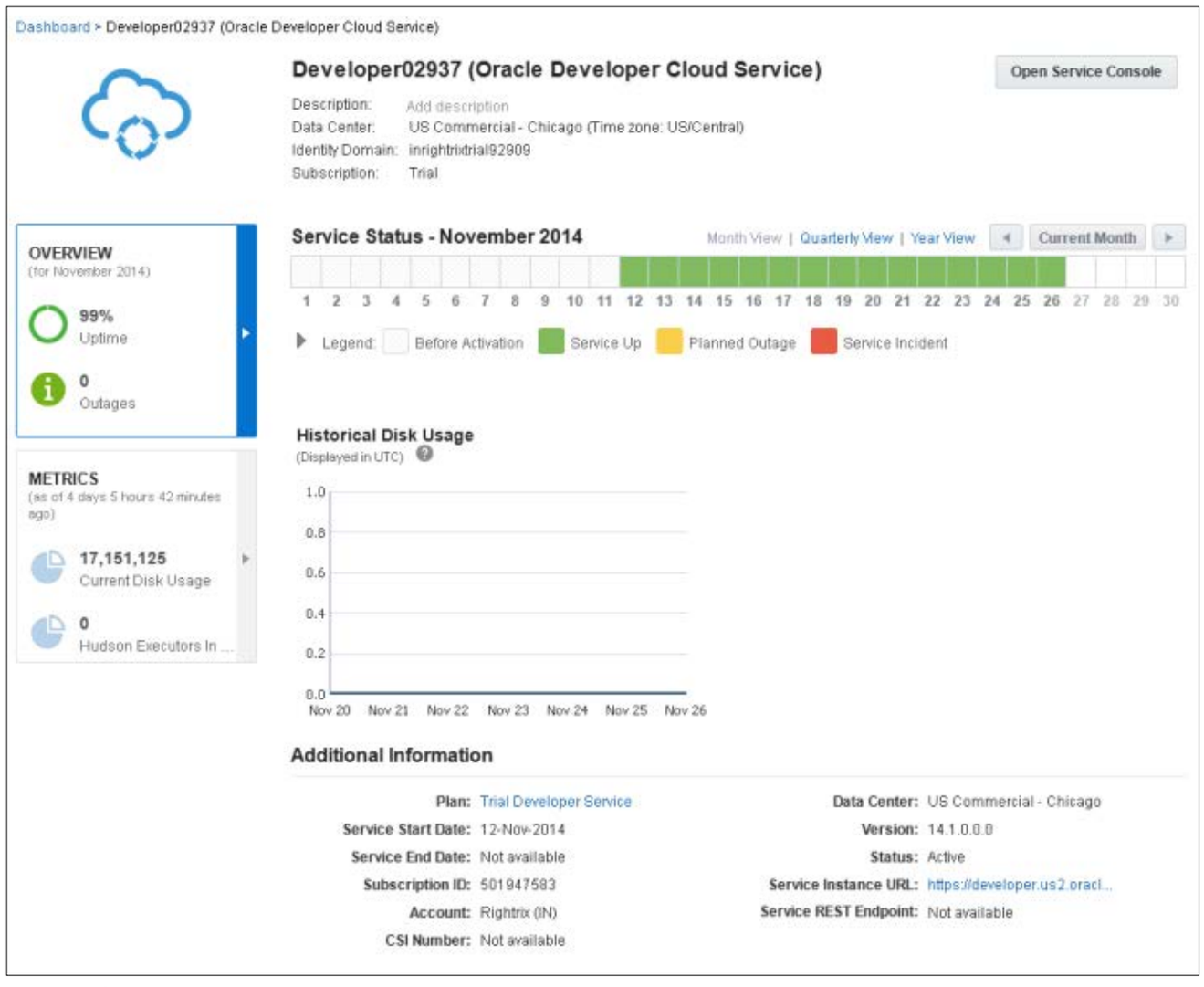


Figure 2

Figure 3

operate out of their familiar IDE than a web interface. So it is vital that any cloud development environment provides integration with popular IDEs. Oracle Developer Cloud Service offers integration with NetBeans, Eclipse, and Oracle JDeveloper, so you can stay in your IDE and create tasks, commit code, and perform team activities. You can create an Oracle Developer Cloud Service project right from your IDE, or you can first create it using the Oracle Developer Cloud Service web console and later integrate it with your IDE.

Let's now look at how you could easily integrate Oracle Developer Cloud Service with NetBeans IDE so that you could perform most of your Oracle Developer Cloud Service tasks via NetBeans.

You first need to install the Oracle Developer Cloud Service plugin. Select **Tools -> Plugins** from the NetBeans menu. Then click the **Available Plugins** tab and search for *developer cloud* to get the screen shown in **Figure 4**. Install the Oracle Developer Cloud Service plugin. NetBeans will also download and install other plugins that the Oracle Developer Cloud Service plugin depends on.

Note: I would recommend that you also look at integrating Oracle Java Cloud Service with NetBeans IDE, as discussed in my earlier *Java Magazine* article.

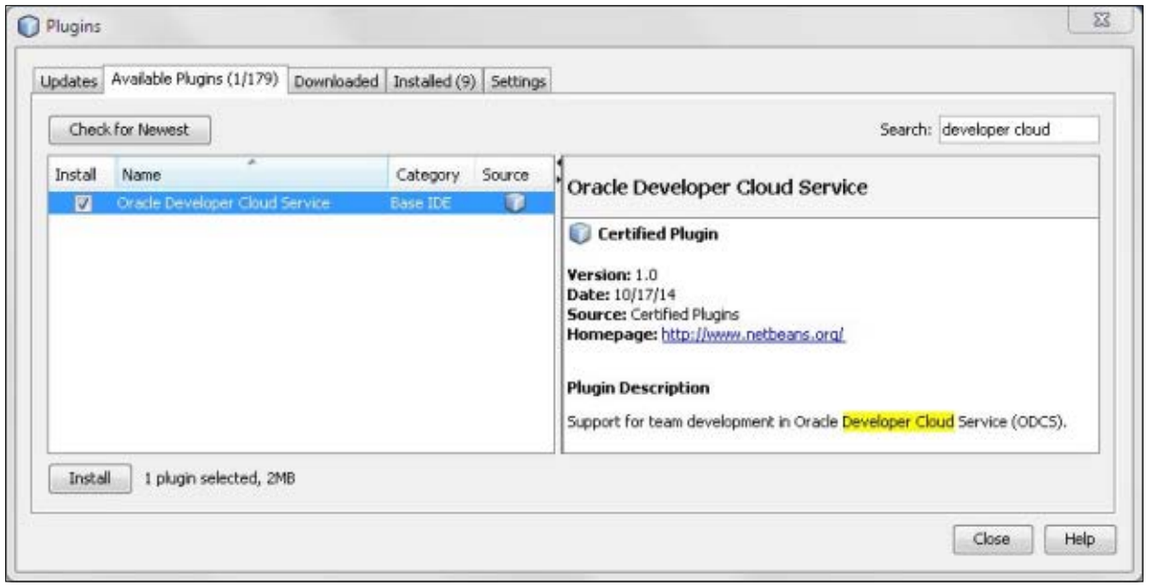


Figure 4

After the plugin is installed, select **Team -> Team Server -> Add Team Server** from the menu to add Oracle Developer Cloud Service as a team server in NetBeans. Now enter the service instance URL (shown in **Figure 2**) into the URL field shown in **Figure 5**. Note that your URL will vary from the one seen in these figures. Also, enter "Java Mag" into the Name field.

After your server is added, you next need to log in to the server by selecting **Team -> Team Server -> Login** from the menu to open the dialog box shown in **Figure 6**.

Your IDE is now set up to use Oracle Developer Cloud Service and the Java Magazine project that we previously created. Select **Team -> Team Server -> Java Mag -> Open Project**. Click the **Search** button and you should see the screen shown in **Figure 7**.

Click **Open from ODCS Server** and you will now see that the Team window displays your project builds, tasks, and sources, as shown in **Figure 8**. Your NetBeans IDE is now set up and ready to use Oracle Developer Cloud Service as the team server.

You can now get your source code from the Oracle Developer Cloud Service team server by double-clicking the Sources link shown in **Figure 8**, followed by clicking the **Get From Team Server** button shown in **Figure 9**.

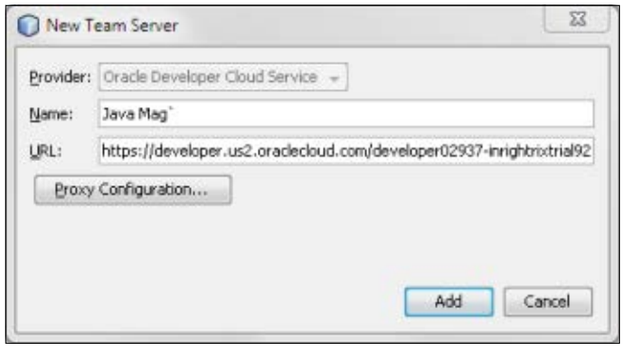


Figure 5

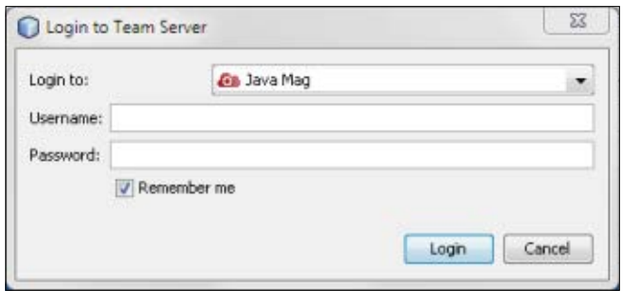


Figure 6

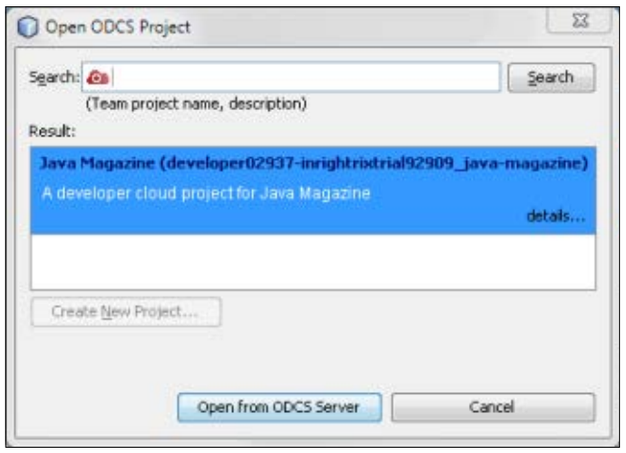


Figure 7

Our Java Magazine project on Oracle Developer Cloud Service currently has no source files, so we could push an existing NetBeans project to the Oracle Developer Cloud Service repository by selecting **Team -> Remote -> Push** from

the menu, which will display the screen shown in **Figure 10**.

If you do not have a local Git repository for your NetBeans project, you would need to first initialize the local repository by using **Team -> Git -> Initialize Repository** for that project.

After your IDE integration with Oracle Developer Cloud Service is set up, you should soon find yourself using the cloud-based Oracle Developer Cloud Service environment with almost as much ease as a local development environment.

Other Considerations

While it's true that development environment clouds leverage mainstream ideas and open source projects such Git, Maven, Hudson, and so on, you still need to consider that the intermediate development environment is vendor-specific, and so there is the possibility of getting locked into a service. Also, as with many other cloud-based solutions, there isn't much standardization happening yet for cloud-based development environments.

Figure 8

Figure 9

The following are some of the other questions you need to think about when considering a cloud-based development environment:

- Is the development environment capable of being extended?
- Will it be responsive enough to your requirements?
- Does the environment provide requisite APIs for you to integrate with any other products that you use?
- Do you need an option for on-premises deployment?
- Would you want a choice of cloud vendors to deploy to?
- Last but not least, is the UI slick and efficient enough for developers to love it and adopt it wholeheartedly?

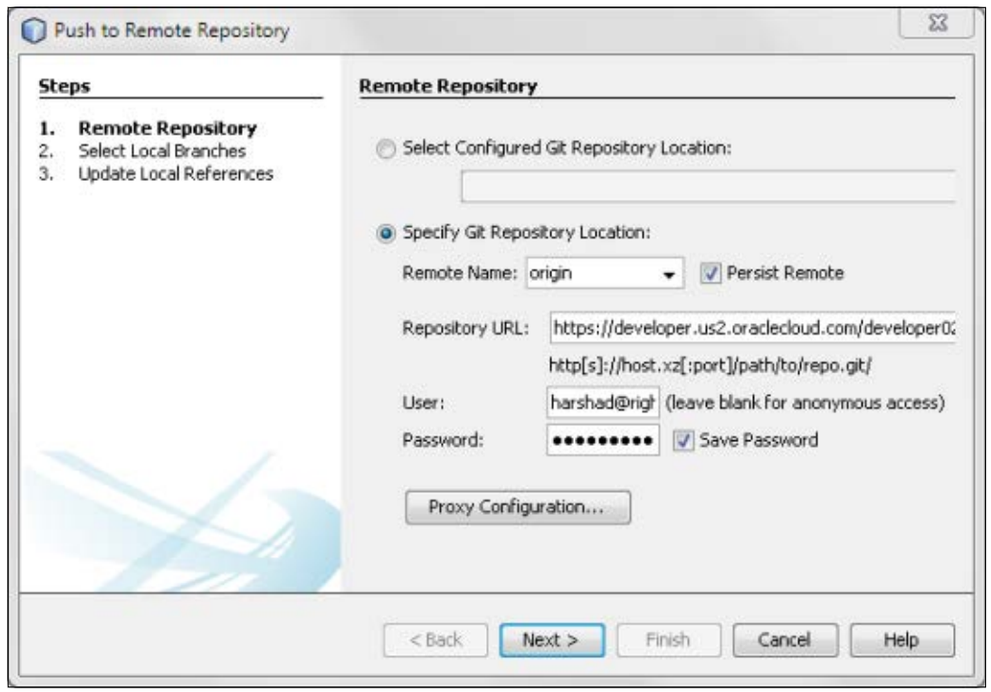


Figure 10

Conclusion

There are good reasons why companies have traditionally looked to manage their own development environment. However, considering the significant benefits of modern cloud-based development environments, they definitely merit a close look.

Oracle Developer Cloud Service is a landmark offering and a sign of things to come, as many more vendors seek to tap the market for "development environment as a service." It is built to cater to the needs of the modern development environment. In addition to offering integration with NetBeans, Eclipse IDE, and Oracle JDeveloper, Oracle Developer Cloud Service enables you to provision your development environment quickly and run your entire development environment in the cloud. `</article>`

MORE ON TOPIC:

Cloud Computing

LEARN MORE

- Oracle Developer Cloud Service website
- Oracle Developer Cloud Service resources

BERT ERTMAN

BIO

Building Modular Cloud Applications in Java

Add, replace, or remove modules while keeping the architecture intact and maintainable.

Cloud Computing

PHOTOGRAPH BY TON HENDRIKS

For the past couple of years, my Luminis colleagues and I have been building cloud applications that conform to a dynamic services architecture using an OSGi-based development stack. OSGi? You are probably thinking, why on earth would I need that? The reason is that we want to have modular applications, because they turn out to be a great fit for the cloud.

Some of the customers that we are working for have adopted the cloud, because they want to be a "first mover" in a new or changing market. Also, they want to be able to "pay as they go" for their infrastructure, and the requirements of the applications that we build for them tend to change at a fast pace.

Dealing with changes in a codebase is not trivial. Instead of designing the perfect system from scratch (which is just impossible), we decided to split its architecture into small, disposable, interconnecting modules. This enables parts of an application to be added, replaced, or removed at will, while the architecture is kept intact and maintainable. It also provides a mechanism to deal with change without breaking the application while refactoring the architecture.

Note: The source code for the sample application described in this article is available here.

Background on Modularity

Modularity is not a new concept; it has been around for a long time. The term *modularity* was first coined in the late 1960s by Professor Edsger W. Dijkstra in one of his manuscripts.

Unfortunately, the Java platform itself does not have a module system available out of the box. Project Jigsaw is in the works, but it isn't available for production usage yet. Because we needed a modularity solution now, OSGi was chosen, which offers the benefits of having a dynamic modular runtime on top of the Java Virtual Machine (JVM). OSGi has been a proven modularity solution for more than 10 years.

Avoid Coupling, Promote Cohesion

Modularity is about isolating classes that have a (tight) form of coupling. You can isolate them by placing them into a module (or *bundle*, as OSGi calls them), and then you can define strict dependencies on interfaces and classes that you import from other modules (*cohesion*).

Inside these modules, you define services that are registered at a central *service registry*. Via the registry, you obtain references to services. This mechanism, in practice, is somewhat similar to dependency injection frameworks such as Spring or Contexts and Dependency Injection (CDI).

Figure 1 shows modules that depend on other modules, being explicit about what they share and hide. **Figure 2** shows a service registry for implementing inversion of control.

Defining an API

In practice, when designing a modular infrastructure, you usually start off by defining an API module. Other modules in the system can depend on this module, while you can

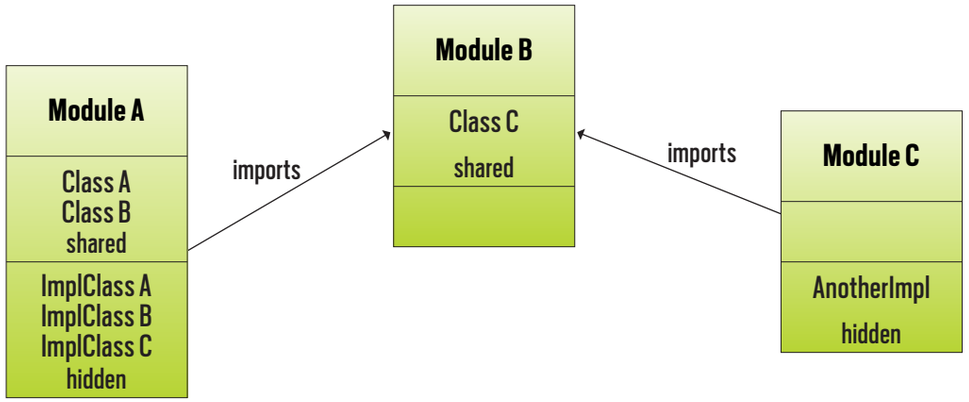


Figure 1

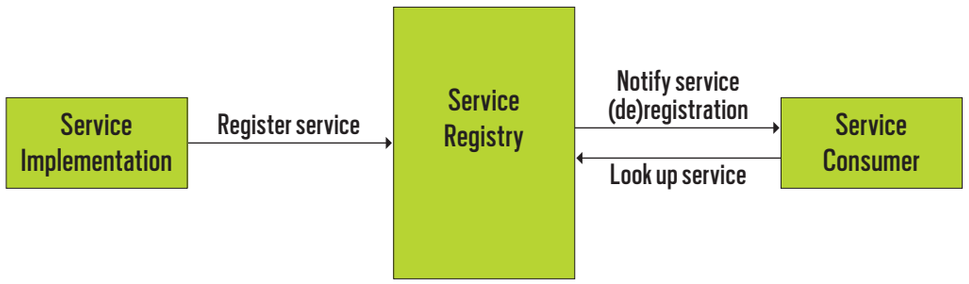


Figure 2

keep changing the implementation of the API, which resides in a different module.

Listing 1 shows an Agenda interface, and **Listing 2** shows a Conference domain class.

Creating an Implementation

Next, I create a simple implementation for our service. Implementation classes typically implement something from the API module and use some of the domain classes or value objects defined in that module. Internally they may use other classes that are not exposed to the outside world, preventing coupling at this level. **Listing 3** shows a simple implementation of the Agenda interface.

Wiring Things Up

Our development stack is based on Java SE 8, and the Apache Felix OSGi runtime is our deployment environment. Our production systems contain a basic Linux image with just the JVM and Felix on them. Dependencies on OSGi are minimized by abstracting them away in so-called *activator* classes.

An activator uses the Felix Dependency Manager framework to register services with the OSGi service registry and to declare dependencies on other services. Felix Dependency Manager is just one of the many choices available to handle dependency management.

The amount of actual OSGi knowledge that you need to start writing applications is very limited. It becomes even less if you use the right set of tools. Our developers use Eclipse with the Bndtools plugin. Bndtools does much of the OSGi boilerplating, for example, generating bundles and the associated OSGi metadata.

Bundles, by the way, are just plain JAR files. Bundles are special only because the JAR files contain metadata that describes public and private "compartments" in which shared and internal classes reside. Metadata also expresses naming, dependencies, and versioning information. **Listing 4** shows how Activator publishes the service as the Agenda API type.

Not Reinventing the Wheel

OSGi sometimes has a reputation for forcing you to reinvent the wheel for everything you want to use with it. It is true that not everything you are used to in a traditional development stack—such as Java EE—is automatically available

LISTING 1 LISTING 2 LISTING 3

```
package agenda.api;

public interface Agenda {
  void addConference(String name, String location);
  List<Conference> listConferences();
}
```

Download all listings in this issue as text

as an OSGi module. You do not want to lose productivity by choosing modularity. Luckily, this need not be the case, because you can use a number of building blocks from the Amdatu project.

Amdatu is an Apache-licensed open source project with an ever-growing collection of OSGi-based cloud components that keep your productivity in a cloud environment on par with that of a traditional development approach. Amdatu features components that deal with REST, persistence, security, multitenancy, templating, logging, e-mail, and much more.

Instead of reinventing the wheel, Amdatu uses existing standards, such as JAX-RS for REST and the Java Persistence API (JPA) for relational database access and, therefore, it wraps certain well-known libraries and frameworks to make them available as modules and services to your application.

Adding a RESTful Service

Cloud applications today are typically services that are consumed over the internet. These services are mostly abstracted as RESTful endpoints. Using Amdatu REST, you can easily create a RESTful resource by adding a class with some familiar JAX-RS annotations.

Under the covers, Amdatu REST uses the Whiteboard pattern to pick up service registrations of RESTful endpoints and publish them using Jetty. This means that I can simply expose a service as a RESTful endpoint by adding an Activator that registers a class with the service registry.

As shown in **Listing 5**, the Agenda RESTful endpoint uses plain JAX-RS. In **Listing 6**, you can see that Activator publishes the RESTful endpoint. **Figure 3** shows the Agenda RESTful endpoint in action.

Figure 3

LISTING 4 LISTING 5 / LISTING 6

```
package agenda.simple;

public class Activator extends DependencyActivatorBase {

  @Override
  public void destroy(BundleContext ctx, DependencyManager dm)
    throws Exception {
  }

  @Override
  public void init(BundleContext ctx, DependencyManager dm)
    throws Exception {
    dm.add(createComponent()
      .setInterface(Agenda.class.getName(), null)
      .setImplementation(SimpleAgendaService.class));
  }
}
```

Download all listings in this issue as text

Persisting to MongoDB

To conclude our simple example application, I can add some persistent storage. A lot of web applications have form-based interactions with their users. These forms can often be expressed as structured documents in the back end. Therefore, a document-oriented storage approach, such as MongoDB, might be a good fit.

Once again, I can use an Amdatu component that exposes MongoDB as a service. On top of that, I can use Jongo to enhance the MongoDB interaction.

Listing 7 shows a service implementation that uses MongoDB for persistent storage. **Listing 8** shows an Activator that publishes the Mongo service. **Figure 4** shows how a conference posted through the REST interface shows up in MongoDB.

Disposable Parts

As I developed the MongoDB implementation, I now have a

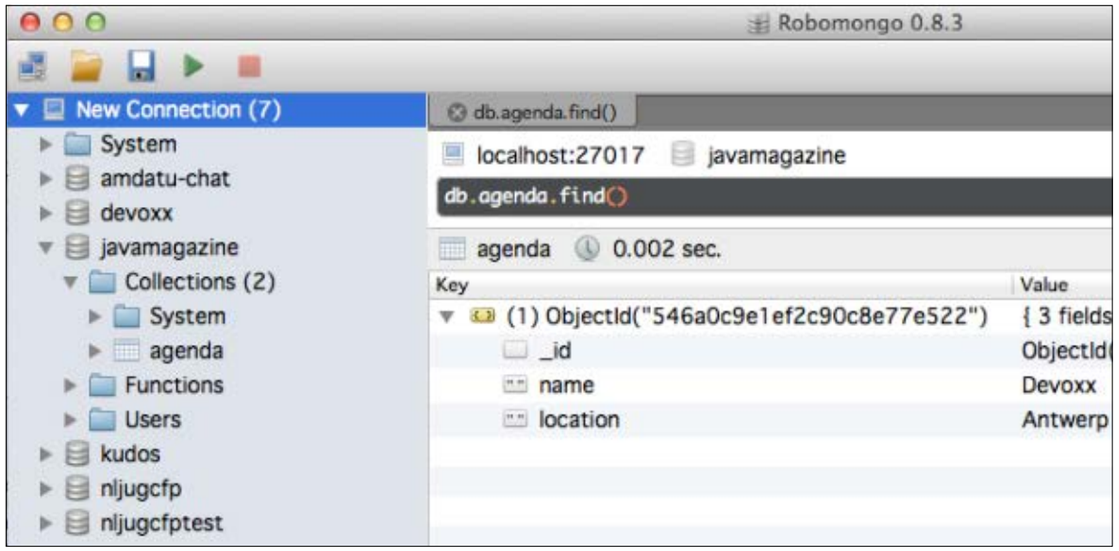


Figure 4

second implementation of the same service available. Multiple service implementations can be very handy in some cases. OSGi provides several capabilities for choosing between available service implementations, such as choosing based on a service priority or based on filter criteria that can be passed to the service registry.

Having another version of a service available might provide a graceful degradation capability if the primary service is down. OSGi will automatically switch service implementations if the primary service becomes (temporarily) unavailable, such as during maintenance or when upgrading to a newer version. If you don't want this capability, you can also choose to just throw the old implementation away. You can do so without breaking anything else in the system, because everything else depends only on the API. It's an example of how disposable components are when you use this modular development approach.

Provisioning and Deployment

By now, you might wonder how we deal with not just a single WAR file at deployment time, but with a number of different modules (JAR files) and their dependencies.

In some of our production systems, the number of JAR files that have to be deployed is close to 300. In order to keep track of them—and their versioning—we use a so-called *provisioning system*: Apache ACE.

ACE can be seen as a giant repository that keeps track of modules and is aware of semantic versioning. ACE allows you to define *feature sets* and *distributions*. Feature sets contain a number of versioned modules, and distributions contain feature sets.

In a cloud environment, you configure nodes to connect to ACE using only a management agent and to download their software distribution after they come online. The cool thing about an application being modular is that you can later make changes to just a single module, and as soon as you upload the module to ACE, ACE will publish just the delta (that is, just the module that has changed) to the running nodes.

We are using this provisioning approach not only for cloud deploy-

LISTING 7 LISTING 8

```
package agenda.mongo;

public class MongoAgendaService implements Agenda {
  private volatile MongoDBService mongoDBService;
  private volatile Jongo jongo;
  private volatile MongoCollection agenda;

  public void start() {
    jongo = new Jongo(mongoDBService.getDB());
    agenda = jongo.getCollection("agenda");
  }

  @Override
  public void addConference(String name, String location) {
    agenda.save(new Conference(name, location));
  }

  @Override
  public List<Conference> listConferences() {
    List<Conference> result = new ArrayList<>();
    agenda.find().as(Conference.class).forEach(result::add);
    return result;
  }
}
```

Download all listings in this issue as text

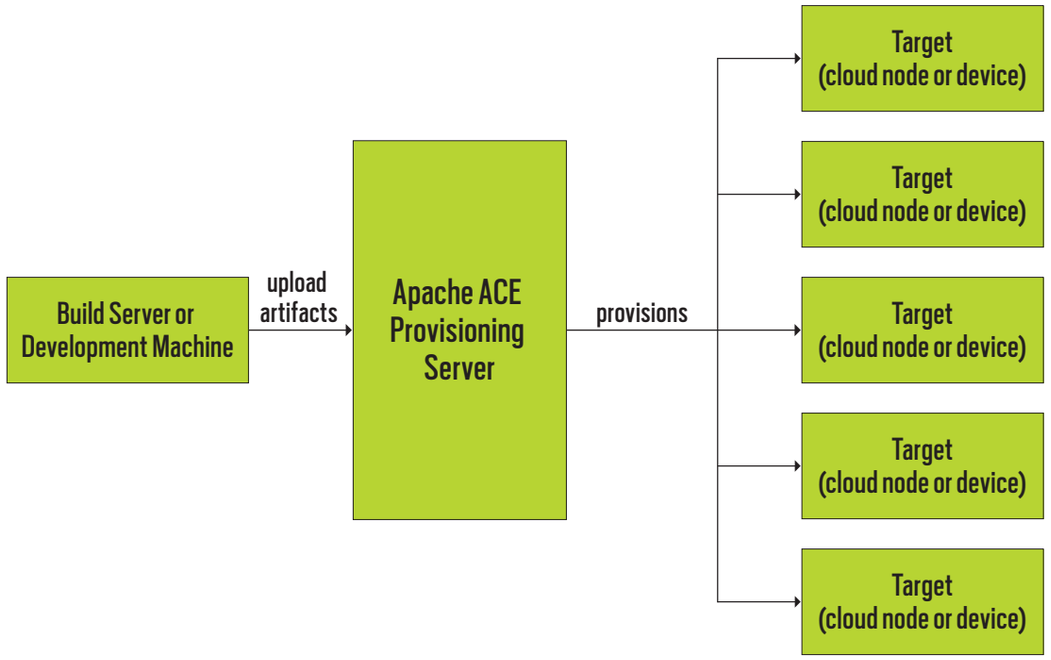


Figure 5

ments but also for devices. Any device capable of running Felix is a fit. **Figure 5** demonstrates Apache ACE as a provisioning server.

More on Deployment

In Java EE environments, you are probably used to the concept of an application server. We decided to go with a lightweight, bare-bones implementation of just Felix.

However, it is perfectly possible to deploy an application to an application server as well, because most application servers today have an internal OSGi-based architecture for modularity. Application servers such as GlassFish and IBM WebSphere offer the ability to deploy OSGi bundles to the application server just as you do for regular WAR or EAR files.

Architecting for Scale

Because the cloud is also about architecting applications for scale, nodes should be kept stateless. Using the load balancer of a cloud infrastructure provider (such as Amazon or Azure), you can keep an eye on the health of the cluster. If a node starts behaving strangely, or if more processing power is needed, the load balancer can start a new node that will automatically connect to ACE, download the latest distribution, and join the cluster.

You can use the same approach to scale down as well, because no one likes to pay for idle servers. This way, you have a truly elastic scaling approach in place.

Having a stateless architecture also means that you need some sort of a datastore to keep track of the data. In practice, we use a number of different solutions (aka *polyglot persistence*). For example, we often use MongoDB for document-driven interactions. When there is a need for relational storage, Luminis uses a JPA-based approach on relational back-end stores. For bigger chunks of binary data, scans, or images, we use BLOB stores. Amdatu provides components to connect to these kinds of stores.

Comparison with Microservices

Finally, I would like to compare our modular approach with what is currently called *microservice architecture*. As you can see from the example code, the end result of our implementation is pretty concise (in terms of lines of code). The result is published as a standalone deployment, runs in a single process, and does one thing well.

Of course there are some additional nonfunctional requirements that should be handled, such as ensuring the resilience of the service and monitoring, but adding some libraries can, for a large part, address those needs. In other words, a modular development approach like the one described in this article suits a microservices-based architecture pretty well, while still providing the ability to update just a small part of the implementation without taking it down.

Conclusion

I hope you have a pretty good idea now how we deal with cloud applications in Java, and why a modular development approach is a good fit for the cloud, especially when used with a provisioning system that is module- and version-aware.

In general, I think modularity is the ultimate agile tool. It helps to deal with a changing codebase, and allows you to worry less about maintenance in the long-term because all service implementations are disposable components that can be added, replaced, or removed without breaking other parts of the application. `</article>`

MORE ON TOPIC:

Cloud Computing

LEARN MORE

- *Building Modular Java Applications in the Cloud Age* (video)
- *Building Modular Cloud Apps with OSGi*
- JavaOne 2013 presentation on building modular applications
- Additional sample applications

ADAM BIEN

BIO

Concurrent Programming with Java EE 7 and Java SE 8

Get the best of both worlds by running Java EE 7 applications on Java SE 8.

Both Java EE 7 and Java SE 8 introduced new support for concurrent and asynchronous programming. New Java SE 8 language features such as method handles, functional interfaces, and lambdas significantly streamline asynchronous execution of methods. New APIs (particularly the Stream API), CompletableFuture, and multiple extensions of the java.util package such as parallel array extensions and streams introduced new ways of code parallelization to Java developers.

PERFECT STORM

Even considered separately, **both Java EE 7 and Java SE 8 bring new APIs, features, and paradigms** to concurrent programming. This article discusses the "perfect storm": running Java EE 7 applications on Java SE 8.

At the same time, the Java EE 7 APIs introduced direct support for asynchronous HTTP processing in JAX-RS 2.0 (JSR 339, The Java API for RESTful Web Services); support for websockets; and even an entirely new API via JSR 236, ConcurrencyUtilities for Java EE.

Even considered separately, both Java EE 7 and Java SE 8 bring new APIs, features, and paradigms to concurrent programming. This article discusses the "perfect storm": running Java EE 7 applications on Java SE 8.

It Started with Java EE 6

Java EE 6 introduced the @Asynchronous annotation for EJB 3.1. Any method denoted with the javax.ejb.Asynchronous annotation is executed asynchronously in a separate thread and transaction. **Listing 1** shows an example of the asynchronous execution of synchronous methods.

Generally, any EJB bean can be turned into a general-purpose java.util.Executor realization. For example, **Listing 2** shows an EJB-based Executor implementation.

A general-purpose Executor accepts java.lang.Runnable implementations and executes them in a manner that is implementation dependent. **Listing 3** shows wrapping the application code with Runnable.

In **Listing 3** the Executor is implemented with an EJB bean, which ensures a transactional execution. Therefore, the method Runnable#run will be executed within a fresh transaction.

With lambdas, the creation of the anonymous Runnable instance can be significantly streamlined:

```
Runnable run = () -> { //... };
```

In fact, the save method can be reduced to a single line, as shown in **Listing 4**, which shows implementing a Runnable "on the fly."

Any method with a matching signature could be used instead of a Runnable implementation. The Runnable interface is a single abstract method (SAM) interface—an interface with only a single abstract method. Any SAM could be directly imple-

PHOTOGRAPH BY THOMAS EINBERGER/ GETTY IMAGES

mented with an instance or static method handle.

Equally well, a method of an injected plain old Java object (POJO) can be passed to the Executor as a Runnable (see **Listing 5**).

Code Pruning with Java EE 7

A custom implementation of the EJB-based Executor comes with a major drawback: There is no standard way to throttle the number of parallel requests or influence the configuration of the underlying thread pool. Java EE 7 with JSR 236, Concurrency Utilities for Java EE, comes with a managed implementation of the Java SE java.util.concurrent.ExecutorService implementation. ExecutorService extends the Executor interface, which makes the custom EJB implementation obsolete.

The ManagedExecutorService shown in **Listing 6** introduces a managed ExecutorService implementation that is an application server's managed resource and, therefore, is injected with the @Resource annotation.

Listing 7 shows String decoration within a POJO. Before being stored, the input is processed by the Normalizer#normalize method.

CODE REDUCTION
With a little help from Java SE 8, we can chain together multiple components and **reduce the code to a single line.** At the same time, Java EE 7 provides serviceability features.

Java SE 8 comes with a set of useful built-in SAMs in the java.util.function package. The Consumer<String> interface can be used as an abstraction of the Storage#store method, and a Function<String,String> can be used as a placeholder for the Normalizer#normalize method, as shown in **Listing 8**.

More Code Annihilation with Java SE 8

Java SE 8 brings an interesting orchestration utility: the java.util.concurrent.CompletableFuture class. The CompletableFuture connects Supplier instances with Functions and Consumers and, thereby, allows the realization of flexible execution pipelines. The example from **Listing 8** can be greatly simplified using CompletableFuture. **Listing 9** shows connecting Storage to Normalizer with CompletableFuture.

The assignment to the Function and Consumer interfaces was used only for demonstration purposes.

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

```
@Stateless
public class StorageService {
  @Asynchronous
  public void save(String content) {
    //heavy lifting
    System.out.println("Storing " + content);
  }
}
```

Download all listings in this issue as text

Unfortunately, the code in **Listing 9** is not executed by the managed thread pools of the application server; rather, it is executed by the ForkJoinPool instance known as the "common pool": ForkJoinPool.commonPool().

Thankfully, all method names ending with the "Async" suffix accept an Executor instance as a second parameter. ManagedExecutorService is an Executor that can be used to execute the pipeline by a managed application server's threads. **Listing 10** shows passing ManagedExecutorService as an Executor and using method handles as SAMs.

With a little help from Java SE 8, we can chain together multiple components and reduce the code to a single line. At the same time, Java EE 7 provides serviceability features such as management, monitoring, and configuration capabilities for ManagedExecutorService.

Exceptions Happen

All exceptions that occur in the pipeline in **Listing 10** are going to be silently swallowed. The "no news is bad news" behavior is a good starting point, but it is not acceptable for the majority of use cases. The enclosing StorageService#save method might finish before the pipeline processes the tasks. CompletableFuture provides both blocking and asynchronous excep-

tion handling. The Future#get method blocks the caller and returns either the return value or ExecutionException, which contains the actual cause. **Listing 11** shows an example of blocking for exceptions.

Although the get() method in **Listing 11** exposes the exception, it also blocks the calling method until the CompletableFuture completes all tasks. **Listing 12** shows asynchronous exception handling. The exceptionally() method, which is called asynchronously in the event of an exception, expects a Function<Throwable,[RETURN_VALUE]> as a parameter.

Interestingly, the exceptionally method could be applied to any stage with a varying RETURN_VALUE type. For example, right after the supplyAsync and the thenApplyAsync, the second generic parameter would be a String. Functions are SAMs and can be easily implemented with method handles. Moving the error-handling code into a dedicated handler streamlines the code again. **Listing 13** shows exception handling in a dedicated handler.

Because exceptions happen asynchronously, they cannot be handled in any meaningful way within the StorageService class, and they can be completely extracted into a dedicated handler. The save method is implemented in a "fire and forget" fashion—unsurprisingly, the error-handling strategy is identical to message-driven beans (MDBs) and Java Message Service (JMS).

Errors in asynchronous code cannot be propagated to the caller, so usually they are rerouted to dedicated JMS queues, which are often called *dead letter queues*. The same mechanism can be applied to CompletableFuture without even using JMS. Plain Contexts and Dependency Injection (CDI) events are perfectly suitable for handling asynchronous exceptions. **Listing 14** shows sending exceptions as events.

In addition, the javax.enterprise.event.Event#fire method is a generic Consumer<Object>. With this method, a direct reference to Storage could be entirely replaced by another injected Event instance.

POWER OF TWO

Compared with Java SE 8, the impact of Java EE 7 on an application's architecture is negligible. Really interesting things happen if we combine both: Java EE 7 productivity with Java SE 8 power.

LISTING 7 LISTING 8 / LISTING 9 / LISTING 10 / LISTING 11 / LISTING 12

```
public class Normalizer {
  public String normalize(String input) {
    return "# " + input;
  }
}
//...
 @Inject
  Normalizer normalizer;

  public void save(String input) {
    String content = normalizer.normalize(input);
    executor.execute(() -> storage.store(content));
  }
```

Download all listings in this issue as text

Listing 15 shows using CDI events as a Consumer<String>.

The Storage class remains almost untouched; only the String parameter in the save method needs to be denoted with the @Observes annotation. **Listing 16** shows converting the Storage class into a CDI listener.

Asynchronous JAX-RS 2.0

So far, the JAX-RS resource looks rather unspectacular. Both the @POST and @GET methods in **Listing 17** are performed synchronously.

JAX-RS 2.0 introduced an AsyncResponse parameter for asynchronous request handling. With AsyncResponse denoted with the @Suspended annotation, the resource's method can quit before the request is completed. This approach decouples the connection-handling thread from the actual business logic processing. **Listing 18** shows an example of such asynchronous request processing.

The waiting request can be resumed by calling the AsyncResponse#resume method. It is a good practice to separate the communication protocol from the business logic implementation and, therefore, it is a bad idea to expose the AsyncResponse to the business logic. Hence, the business logic is implemented behind the boundary, and you will have to send the AsyncResponse to the

business logic without revealing its existence.

Prior to Java SE 8, you would usually encapsulate the AsyncResponse within a custom wrapper, but with Java SE 8, we can just send the interesting AsyncResponse#resume method as a Consumer<String>. No additional abstractions or interfaces are required for a tight integration of the AsyncResponse with the business logic.

Also, the @POST method can be reworked to get rid of the reference to the StorageService entirely. The StorageResource#store method sends the input parameter to the back end, which can be easily abstracted with a Supplier<String>. The parameter is instantaneously converted into a Supplier using lambda expressions and sent as an ordinary CDI event. Using the "native" Java SE 8 interfaces already in the JAX-RS resource further simplifies the StorageService code. **Listing 19** shows asynchronous String delivery.

Listing 19, the getContent method, uses the Storage#retrieve method as a Supplier<String> and connects the output with the Consumer<Object> as input. Also, in this case, a CompletableFuture and the injected ManagedExecutorService are used as a pipeline.

GOOD MONITOR

Asynchronous applications are **harder to monitor and debug** than their synchronous counterparts. Interceptors are perfectly suitable to monitor the individual tasks as well as the boundary.

Monitoring with Java EE 7

Asynchronous applications are harder to monitor and debug than their synchronous counterparts. Interceptors, introduced in Java EE 5, are perfectly suitable to monitor the individual tasks (controls) as well as the boundary. **Listing 20** shows a simplistic performance-measuring interceptor.

A single annotation (@Interceptors(Monitor.class)) suffices, and all public methods of the annotated class get intercepted and monitored. The Monitor interceptor in **Listing 20** relies on System.out.println for logging; in the real world, you would instead again use CDI events to publish the monitoring data to a central place for further analysis.

Deleting Some JPA Code with Java SE 8

With the Java Persistence API (JPA), the Store class could be easily extended with persistence capabilities. **Listing 21** shows persisting content using "stock" JPA 2.1 code. The content String is stored for each user separately.

Let's compute some statistics, for example, the average String size per user. Because such computations could be CPU-intensive, we would like to process them in parallel. JPA queries are returned with streamable Java SE 8 collections. Java SE 8 collections open the door for in-memory processing and analytics with plain Java SE 8. **Listing 22** shows creating Store statistics on the fly.

Given that the whole result list fits into RAM (which gets cheaper each day), you can easily implement, test, and perform queries with Java SE 8 lambdas on live JPA entities, as in **Listing 22**. Java SE 8 introduced various Collector statistics utilities, so you can get interesting calculations without any further coding. The IntSummaryStatistics utility class can be easily exposed as a JsonObject. For example,

LISTING 13 | LISTING 14 | LISTING 15 | LISTING 16 | LISTING 17 | LISTING 18

```
public void save(String input) {
  CompletableFuture.supplyAsync(() -> input, executor).
      thenApplyAsync(normalizer::normalize, executor).
      thenAcceptAsync(storage::store, executor).
      exceptionally(this::handle);
}

public Void handle(Throwable error) {
  System.out.println("error = " + error);
  return null;
}
```

Download all listings in this issue as text

Listing 23 shows converting IntSummaryStatistics into a JsonObject.

Parallel streams come with thread-management overhead and do not perform well for simplistic use cases. Also, in the majority of use cases, **Listing 22** will probably perform better with a single-threaded stream rather than by using the corresponding parallelStream. In addition, the asynchronous parallelStream is not executed by the managed threads of the application server, but instead by threads provided by the ForkJoinPool. The number of available fork/join threads depends on the number of available CPUs, but can be configured with the java.util.concurrent.ForkJoinPool.common.parallelism system property. The application server remains unaware of the "wildly" created fork/join threads, which can cause performance bottlenecks or even robustness issues due to too many threads.

QUITE A DRAW

Java EE 7 draws lots of attention; "Java EE" in a title is sufficient to cause overcrowded conference rooms and popular blog posts.

Conclusion

Java EE 7 draws lots of attention; "Java EE" in a title is sufficient to cause overcrowded conference rooms and popular blog posts. However, compared with Java SE 8, the impact of Java EE 7 on an application's architecture is negligible. Really interesting things happen if we combine both: Java EE 7 productivity with Java SE 8 power.

This article was intended to spark some ideas for leaner code and less cargo cult programming. At the same time, the sample code presented in this article is vastly simplified. We persisted a String with the highest possible use of Java SE 8 and Java EE 7 APIs. In the real world, you should instead concentrate on the core business logic, rather than on cool—but for your customers, meaningless—technology. **</article>**

LEARN MORE

- Sample project for this article
- JSR 346, Contexts and Dependency Injection for Java EE 1.1
- "Going Asynchronous With CompletableFuture and Java EE 7" screencast

LISTING 19 / LISTING 20 / LISTING 21 / LISTING 22 / LISTING 23

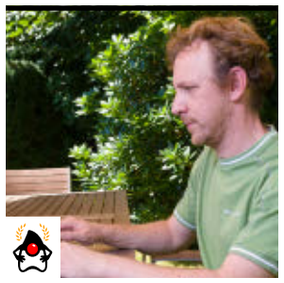
```
@Stateless
public class StorageService {
  @Resource
  ManagedExecutorService executor;
  @Inject
  Normalizer normalizer;
  @Inject
  Event<Throwable> exceptionHandler;
  @Inject
  Event<String> sink;
  @Inject
  Storage storage;

  public void save(@Observes Supplier<String> supplier) {
    CompletableFuture.supplyAsync(supplier, executor).
        thenApplyAsync(normalizer::normalize, executor).
        thenAcceptAsync(this.sink::fire, executor).
        exceptionally(this::handle);
  }

  public Void handle(Throwable error) {
    exceptionHandler.fire(error);
    return null;
  }

  public void getContent(
@Observes Consumer<Object> listener) {
    CompletableFuture.supplyAsync(
        thenAcceptAsync(listener, executor).
        exceptionally(this::handle);
  }
}
```

Download all listings in this issue as text



JOHAN VOS

BIO

Cloud-Based Monitoring of IoT Devices

A performant and scalable cloud-based monitoring system for collecting data from embedded devices

The number of embedded devices that are connected to the internet—the Internet of Things (IoT)—is growing every day. Many of these devices need an internet connection to retrieve data from and send data to a variety of back-end services. But the internet connection often is also required in order to manage the devices and the software they are running.

In order for a device to be manageable, it should at least be possible to monitor it. An operator, or the person who is responsible for the maintenance of the device, wants to know the current state of a device, its history, its current operations, and so on. With the growing number of connected devices, monitoring these devices becomes a huge but important challenge. In this article, we will explore a use case where cloud-based monitoring of connected devices allows the operation of a large number of kiosk systems.

The Use Case

CultuurNet Vlaanderen is a government organization in Belgium. At the request of the Flemish authorities, this organization tries to increase public enthusiasm for culture. In 2012, CultuurNet launched the Uitpas project, a system based on a near field communication (NFC) card that allows its users to earn points when attending cultural events and to exchange points for rewards. Users can scan their cards at kiosk systems (see **Figure 1**), which are built around a Raspberry Pi connected to an NFC reader and an LCD screen.

A kiosk system is also called a check-in device (CID), because the original goal was to "check in" a user for a cultural activity. Initially, Uitpas was rolled out in three cities only, with about 30 CIDs spread around cultural locations in those cities, for example, near theaters, cinemas, museums, and so on. Those CIDs were operated by CultuurNet on behalf of the three cities.

The success of Uitpas in the three cities triggered an interest by more cities, so more CIDs were put into production, along with a growing number of operational entities. Each city is now expected to manage its own devices and operations.

Clearly, management can't be done manually anymore. Also, the need for distributed management (each city should be able to monitor and fix its own devices) leads to operational challenges. In order to address these challenges, CultuurNet is now using a cloud-based monitoring system that monitors all registered CIDs and provides access to different operators. Each operator—typically, an IT manager employed by the city—can monitor and man-

Figure 1

Cloud Computing

PHOTOGRAPH BY TON HENDRIKS

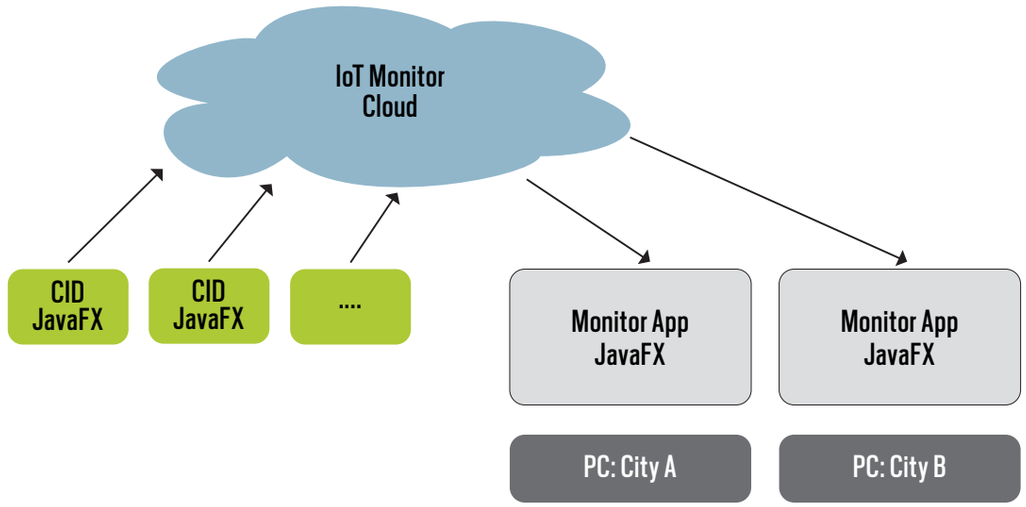


Figure 2

age only the CIDs for that manager's city.

The solution is powered end to end by Java. The CIDs run JavaFX on a Raspberry Pi. They send their monitoring information to the IoT Monitor Cloud—the central piece of the architecture, which runs a Java EE 7 implementation (GlassFish). The client monitoring application is, again, a JavaFX application. **Figure 2** shows this setup.

The different components all use Java, but in different environments. The IoT Monitor Cloud collects and stores all information from all CIDs. This component uses Java EE 7 APIs and runs in GlassFish instances in the Amazon EC2 cloud. It uses Amazon DynamoDB for storage capabilities and the open source elasticsearch engine for search capabilities.

CIDs to Cloud

The CIDs send monitoring data about functional events, as well as logging data, to the IoT Monitor Cloud. Because logging data is easily obtained in a Java application, this functionality can be used in any application. The CID application internally uses the logger shown in **Listing 1**.

All system-related events (for example, a card being scanned, a network request being made, a network request timing out) are logged using standard Java logging commands, for example:

```
LOGGER.log(Level.SEVERE,
"Could not read card.");
```

A specific MonitorHandler, which is an extension of LogHandler, is created and added to the created logger, as shown in **Listing 2**. By doing so, the monitorHandler will be notified about all entries that are logged on the LOGGER instance.

The MonitorHandler itself is defined as shown in **Listing 3**. Whenever something is logged in the application by calling LOGGER .log(...), the publish method on the MonitorHandler is called, and the log message is provided with additional information such as a time stamp, thread information, the method, and so on.

The MonitorHandler will send the log information to the IoT Monitor Cloud using DataFX, which is a JavaFX-based framework that brings enterprise functionality to JavaFX. One of its components is the DataSources component, which facilitates REST-based communication with back-end systems. DataFX respects the JavaFX threading model and does its work on a background thread, using the JavaFX Application thread to report back to the application.

The code snippet in **Listing 4** shows how DataFX executes a REST request from an embedded device to the IoT Monitor Cloud, which provides an endpoint at the fictive address http://iotmonitor.cloud. The actual request contains more form parameters, but for readability those are omitted.

LISTING 1 LISTING 2 LISTING 3 LISTING 4

```
public static final Logger LOGGER =
Logger.getLogger("be.uitpas.pi");
```

Download all listings in this issue as text

As can be seen from the code, a RestSource is created for making an HTTP connection to an endpoint, thereby providing typical information about the path, the parameters, and connection settings such as the timeout value. The RestSource is then passed to an ObjectDataProvider, which makes the request and puts the result into a property named answer. An ExecutorService instance is then passed to the DataFX ObjectDataProvider, because it can be expected that on some occasions, lots of logging information will have to be sent. Using a single thread for all communication would potentially delay the transmissions, whereas using one thread for each message might lead to too many resources being consumed. Hence, an ExecutorService is provided with typically five threads.

A deeper discussion of DataFX is outside the scope of this article. See http://datafx.io for more information.

One of the most common issues with connected devices is the loss of connectivity. Clearly, sending a log message about lost connectivity is not going to work. In this case, the MonitorHandler will store all log messages to a file system. Once the connection is restored, the messages will be sent to the IoT Monitor Cloud.

MONITOR IT

The Java EE 7 APIs allow you to develop a cloud-based monitoring system that is performant and scalable while addressing the needs of clients (for example, by providing support for WebSocket).

JavaFX Monitor Client Application

The JavaFX Monitor client application connects to the IoT Monitor Cloud and visualizes the monitoring information that the operator is allowed to see. The IoT Monitor Cloud maintains a list of registered operators, and associates them with one or more card systems. A card system is a group of CIDs that belong together, typically all CIDs in a specific city.

Authentication is, therefore, a crucial component in the IoT Monitor Cloud setup. **Figure 3** shows the login screen of the JavaFX Monitor client.

Once the operator is authenticated, the monitoring console is shown (see **Figure 4**).

The top-left corner of the console shows the list of card systems the operator has access to, and the

Figure 3

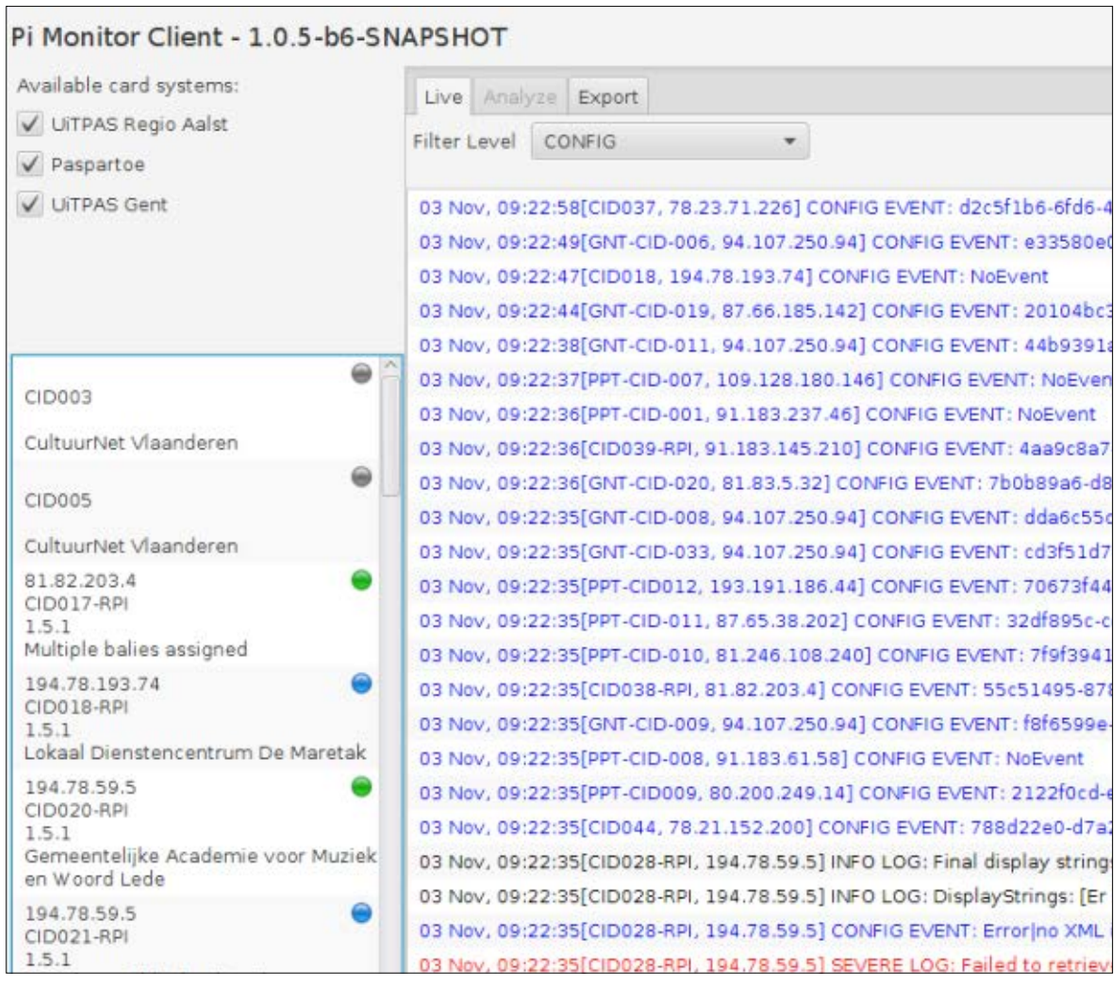


Figure 4

CIDs registered with those card systems are shown below. The main part of the console shows the monitoring data originating from the different CIDs that the operator is managing.

Typically, there is a huge amount of data to be visualized. Therefore, the application contains a number of selectors and filters that allow the operator to select a particular part of the data, for example:

- A time selector, for showing only information within a specific time range.
- A level selector, for selecting messages of a specific level or a range of levels.
- A CID selector, for selecting one or more CIDs for which to inspect logs. Clicking a specific CID in the list on the left selects the specified CID and shows its monitoring data in the list. If no specific CID has been selected, data from all CIDs is shown.

When analyzing data, operators typically modify the selectors frequently. If all changes to selectors would lead to calls to the back end, the application would not be very responsive. However, JavaFX in combination with Java 8 features allows operators to manipulate the data set on the client. When the application is started, some data is retrieved. If more data is needed, it will be loaded into the application, and data that is no longer needed can be garbage collected. This allows the application to take advantage of the memory and processing capabilities of the client system it is running on. Typically, the JavaFX Monitor application has a lot more data entries in its memory than it visualizes in the list.

The list component in the main part of the console is defined as follows:

```
private ListView<Event>
eventsListView
```

The items that are to be rendered in this ListView are kept in a JavaFX SortedList named sortedEvents. This list is associated with the ListView as follows:

```
eventsListView.setItems(
sortedEvents);
```

All data for monitoring items that are retrieved from the IoT Monitor Cloud are captured in an ObservableList of type Event that is named events:

```
ObservableList<Event> events;
```

New events are loaded from the back end using DataFX with code that is similar to the code in **Listing 4**, or they are sent from the IoT Monitor Cloud to the JavaFX Monitor application over a WebSocket, which will be discussed shortly. In both cases, the fact that events are stored in an ObservableList allows the ListView that renders the events to take all data into account—including events that are added after the ListView has been created.

Note that the raw events are stored in an ObservableList<Event> named events, whereas the events that are passed to the ListView are stored in a SortedList<Event> named sortedEvents. There is a relationship between those two lists. First, the raw list of events is filtered to contain only those events that match the different selectors. This is done using the Java 8 Stream API, which provides filtering functionality. We have an intermediate field named filteredEvents of type FilteredList<Event> that is initially defined as shown in **Listing 5**.

The FilteredList class was added in JavaFX 8, and it provides an extremely powerful approach for maintaining a list of elements based on an original list, but taking into account filter criteria that are specified as a predicate.

From this initialization, it is clear that initially, all raw events are considered to be included in the filteredEvents list. Whenever a selector changes, the filtered predicate is changed:

```
filteredEvents.setPredicate(
validEvent())
```

validEvent() returns a predicate that checks whether a given candidate event should be contained in the filteredList.

In our case, the boundary conditions imposed by the three selectors mentioned previously must be satisfied before a candidate event is accepted. This is shown in the implementation of the validEvent() method in **Listing 6**.

The return statement in this method contains three methods that each returns a predicate by itself. We will examine only the code for the isShowLevel() predicate

LISTING 5 LISTING 6 / LISTING 7

```
FilteredList<Event> filteredEvents = events.filtered (e → true);
```

Download all listings in this issue as text

(see **Listing 7**); however, it should be clear that the other functions are constructed in the same way—but they are slightly more complex. This predicate will check that for a given event e, the log level is at least equal to the level selected by the selector.

The filteredEvents could now be shown in the ListView, but there is no guarantee on the order in which the events would be shown. This can be fixed by having the Event class implement the java.util.Comparable interface, and implement it in such a way that the most-recent events are always shown first. The SortedList class was added in JavaFX 8, and it allows instances of a ListView to be sorted according to their Comparator, which is what we use:

```
SortedList<Event> sortedEvents =
filteredEvents.sorted();
```

It is this SortedList, which extends ObservableList, that is passed to the ListView. Thus, by using some Java 8 functionality, we are able to use a simple approach to render only relevant entries in a sorted way, based on a large set of raw, unsorted data.

WebSocket

The time selector shown in **Figure 4** shows a tab labeled "Live." This allows real-time monitoring.

Real-time monitoring of devices is often very important, because it can prevent issues from being escalated into big problems. It also helps operators to detect problems rather than requiring users to call an operator to report a problem. Therefore, the IoT Monitor Cloud is capable of sending data entries on the fly, as soon as they arrive in the system.

Both the IoT Monitor Cloud as well as the JavaFX Monitor application leverage JSR 356, the Java API for WebSocket. As soon as a user logs in successfully to the JavaFX Monitor application, the application establishes a WebSocket connection to the IoT Monitor Cloud. The DataFX framework we discussed before contains a WebSocket component as well. This component leverages the client portions of JSR 356 and opens a WebSocket connection to the IoT Monitor Cloud. Because the IoT Monitor Cloud is running on top of GlassFish, which is the reference implementation of Java EE 7, it contains an implementation of JSR 356 out of the box. Hence, it is very easy to register a WebSocket endpoint in the IoT Monitor Cloud that will broadcast all new incoming events to the clients that are entitled to read this information.

Conclusion

In this article, we talked about the importance of being able to monitor IoT devices in the field. When a large number of devices is being used, and when those devices generate lots of data, a cloud-based solution is preferred for collecting the data, especially if there are different groups of users with different sets of permissions. The Java EE 7 APIs allow you to develop a cloud-based monitoring system that is performant and scalable while addressing the needs of clients (for example, by providing support for WebSocket).

The visualization of the monitoring data can be achieved using JavaFX. We saw that the Java 8 APIs help in filtering and sorting relevant data based on a number of criteria.

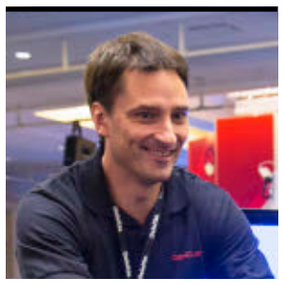
This exemplifies Java everywhere: The embedded devices are running Oracle Java SE Embedded, the IoT Monitor Cloud is running GlassFish 4.1 instances, and the monitoring client is a self-contained application created with JavaFX 8. `</article>`

MORE ON TOPIC:

Cloud Computing

LEARN MORE

- DataFX
- JSR 356



ALEXANDER
BELOKRYLOV

BIO

Get Started with Oracle Java ME Embedded 8 on Microcontrollers

Access peripheral devices using the Device I/O API of Oracle Java ME Embedded.

This article will show you how to get data from a DS1621 sensor—a digital thermometer and thermostat that has an inter-integrated circuit (I^2C) bus interface—by using Oracle Java ME Embedded 8.1, its high-level Device I/O API, and the Freescale FRDM-K64F board, which is based on the ARM Cortex-M4 processor.

In this article, you will learn how to do the following:

- Enable Java capabilities for the Freescale FRDM-K64F board
- Get the protocol information required to write to and read from the DS1621 sensor
- Develop an Oracle Java ME Embedded application that uses the Device I/O API and Java ME SDK tools to access the I^2C digital thermometer

PHOTOGRAPH BY VOLTAIRE YAP

Overview of Oracle Java ME Embedded 8.1

Oracle Java ME Embedded is a Java runtime optimized for resource-constrained embedded devices, for example, devices based on the ARM Cortex-M3 or Cortex-M4 microcontrollers. Using Oracle Java ME Embedded offers several benefits.

First, the "write once, run everywhere" principle—with some restrictions—can be applied to the world of embedded devices by abstracting the hardware and software fragmentation present in today's embedded systems and providing a consistent and feature-rich Java programming layer.

Second, the Application Management System that is integrated into the Oracle Java ME Embedded platform enables deploying, updating, starting, stopping, and removing applications.

Third, you can use the Device I/O API to access peripherals, such as those that use serial peripheral interface (SPI) bus, I^2C, universal asynchronous receiver/transmitter (UART), and general-purpose input/output (GPIO) pins.

Finally, with Oracle Java ME Embedded, you can obtain free and powerful developer tools as well as a set of standard services and APIs for file I/O (JSR 75), wireless messaging (JSR 120), web services (JSR 172), security and trust services (JSR 177), location (JSR 179), XML (JSR 280), JSON, and OAuth 2.0.

The latest version of the Oracle Java ME Embedded platform, released in November 2014, has two significant enhancements. First, it supports ARM Cortex-M3 and Cortex-M4 microcontrollers with very limited amounts of RAM by delivering a port for the Freescale FRDM-K64F board with the mbed OS.

Second, it provides support for the Eclipse integrated development environment (IDE). The Java ME SDK plug-in for Eclipse is based on the recently released Mobile Tools for Java 2.0, which was developed with significant input from Oracle engineers. Now, all the impressive features of the Java ME SDK—such as on-device debugging, memory and network monitors, as well as CPU and memory profiling—are available in Eclipse. To try them, download and install the Java ME SDK 8.1 Eclipse Plugins file.

Moreover, Oracle Java ME Embedded 8.1 provides

updated support for the Raspberry Pi; improved support for two additional Qualcomm Gobi device families; new communication, security, and networking features; as well as improvements for the developer experience, such as tooling over USB, heap analysis, faster communications, and a number of other enhancements and fixes.

Small Platform, Big Opportunities

The Freescale FRDM-K64F board is based on the MK64FN1MOVLL12 multipoint control unit (MCU), which is 120 MHz; has 1 MB of flash memory and 256 KB of RAM; is low-power; and provides crystal-less USB operation in a 100 low-profile quad flat package (LQFP). In addition, its form factor is compatible with the Arduino R3 pin layout.

The Freescale FRDM-K64F board supports many peripherals to enable rapid prototyping. In addition to UART, I²C, SPI, and GPIO interfaces onboard, it has three LEDs and two switches accessible from the Java runtime. Another helpful device integrated into the Freescale FRDM-K64F board is an accelerometer-magnetometer with an I²C interface (FXOS8700Q). You can easily get access to it from a Java application to create a compass application, for instance.

Hardware Requirements

- Freescale FRDM-K64F board
- MicroSD (secure digital) card of any size
- Micro-USB cable
- Ethernet cable and Ethernet connection between the Freescale FRDM-K64F and your computer
- DS1621 digital thermometer with I²C interface

Software Requirements

Java ME SDK. To get all the power of a Java ME developer experience, you need to download and install the Java ME SDK 8.1 bundle. The bundle includes the following:

- Java ME SDK
- NetBeans plugins
- Eclipse plugins

Install the plugins according to the IDE guidelines.

The current version of Java ME SDK supports only Microsoft Windows. Before starting the installation of Java ME SDK, ensure that you have JDK 7 or JDK 8 installed.

A detailed guide for the Java ME SDK can be found at the Java ME documentation portal.

Java ME runtime. Download a bundle with binaries for the Freescale FRDM-K64F board. You should have the following directories:

- /flash
- /lib
- /sd_card
- /util

Copy the contents of the /sd_card directory to your MicroSD card, and then insert the card into your Freescale FRDM-K64F board.

By default, the Freescale FRDM-K64F board is configured as a DHCP client. However, you assign a static IP address to it by editing the corresponding fields in the /java/jwc_properties.ini file on the MicroSD card.

Enable Java Capabilities for the Freescale FRDM-K64F Board

Now, you can check the new flash capabilities provided by Oracle Java ME Embedded 8.1.

1. Go to the Windows system tray and click **Device Manager 8.1**.
2. Click the **Flash** button and select your connected Freescale FRDM-K64F board from the list.
3. Say, "Wow, it's cool to enable mbed devices with Java capabilities!"

As I mentioned earlier, the Freescale FRDM-K64F board is a very convenient platform for prototyping. It has multicolor LEDs, switches, and other peripherals onboard. You can learn how to work with its LEDs and accelerometer-magnetometer from the Oracle Java ME Embedded blog.

Now, it is time to explore the data sheet for the DS1621 sensor, so we can connect the sensor to the Freescale FRDM-K64F board.

Explore the DS1621 Data Sheet

Note: This section is helpful to those who would like to write Java drivers for devices using the Device I/O API.

The DS1621 digital thermometer and thermostat provides 9-bit temperature indicators that show the temperature of the device. It has an I²C interface that is the same as a two-wire interface (TWI), but it has a different name for licensing purposes.

Usually, chip manufacturers provide developers with a document that contains all the necessary information about how to design a circuit and software to use a particular chip. The DS1621 sensor is no exception; you can download its data sheet here.

Figure 1 shows the pin assignment, and **Table 1** provides a pin description.

Important: Check the Freescale FRDM-K64F board's errata document, which has a correction on the I²C connection for early versions of Freescale FRDM-K64F boards.

The FRDM-K64F board will be operating as a master I²C device, and the DS1621 sensor will operate as a slave. All slave devices need to have an address. The address can be set by connecting the A0–A2

pins either to the VDD pin (logic state 1) or the GND pin (logic state 0). In our case, all pins are connected to the VDD pin.

According to the documentation, the DS1621 address consists of seven bits:

- The first four bits are 1001.
- The last three bits are the logic states of pins A2, A1, and A0.

Therefore, in our case, the address is 1001111 or 0x4f.

Page 7 of the data sheet states that the DS1621 sensor can operate in regular mode (100 kHz clock rate) or fast mode (400 kHz clock rate). We will use the regular mode, so the frequency is 100,000 Hz.

The DS1621 sensor needs to be configured to perform temperature measuring by writing the appropriate bits to the configuration register (0xAC). On page 5 of the data sheet, you can find the definition of the configuration bits. Consider the last bit, LSb. If it is set to 0, the DS1621 sensor will continuously measure the temperature and record the measurements to the temperature register. So, the configuration register value should be set to 0x00.

On page 10 of the data sheet, you can find the command set for the sensor. To start a temperature measurement, we need to send the "Start Conversion T" (0xEE) command to the DS1621 sensor to convert a measured temperature into a digital value.

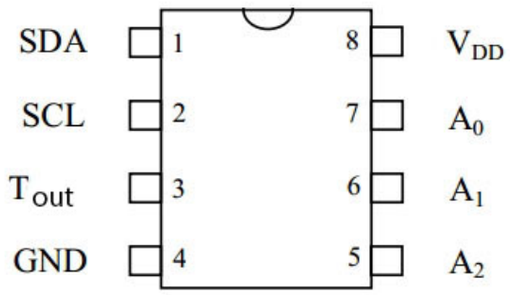


Figure 1

The same page states that to read a temperature measurement, you need to send a "Read Temperature" (0xAA) command to the sensor, and then the sensor will send back two bytes. The first byte is a temperature in Celsius. The leading bit of the second byte is a .5 degree addition. In our case, the temperature resolution is 1 degree; therefore, we need to read only the first byte.

Table 2 summarizes what we have learned in this section.

Develop an Oracle Java ME Embedded Application

Now, we are ready to develop an Oracle Java ME Embedded application to measure temperature.

First, create a new Java ME project in your favorite IDE. As you might know, Oracle Java ME Embedded applications differ from Java SE applications. Oracle Java ME Embedded applications extend the MIDlet class and override two methods: startApp() and destroyApp(). Best practice is to move functionality out of a MIDlet, so let's create a DS1621 class.

To get access to I²C devices from an Oracle Java ME Embedded application, you should use the I2CDevice and I2CDeviceConfig classes from the jdk.dio.i2cbus package of the Device I/O API.

PIN	SYMBOL	DESCRIPTION
1	SDA	DATA INPUT/OUTPUT PIN FOR TWO-WIRE SERIAL COMMUNICATION PORT.
2	SCL	CLOCK INPUT/OUTPUT PIN FOR TWO-WIRE SERIAL COMMUNICATION PORT.
3	TOUT	THERMOSTAT OUTPUT. ACTIVE WHEN TEMPERATURE EXCEEDS A USER-DEFINED HIGH TEMPERATURE (TH); WILL RESET WHEN TEMPERATURE FALLS BELOW A USER-DEFINED LOW TEMPERATURE (TL).
4	GND	GROUND PIN.
5	A2	ADDRESS INPUT PIN.
6	A1	ADDRESS INPUT PIN.
7	A0	ADDRESS INPUT PIN.
8	VDD	SUPPLY VOLTAGE INPUT POWER PIN.

Table 1

ITEM	VALUE
I²C BUS ADDRESS	0X4F
CLOCK RATE	100,000 HZ
VALUE TO WRITE TO THE CONFIGURATION REGISTER (0XAC)	0X00
COMMAND TO START A TEMPERATURE CONVERSION	0XEE
COMMAND TO READ A TEMPERATURE MEASUREMENT	0XAA

Table 2

Initially, we need to create an instance of the I2CDeviceConfig class by using the data from the DS1621 sensor's data sheet, as shown below:

```
cfg = new I2CDeviceConfig(BUS_ID,
                          CFG_ADDRESS,
                          ADDRESS_SIZE,
                          FREQ);
```

Therefore, in our case, we should use the code shown in **Listing 1**. Then, we can create an instance of the I2CDevice class as shown in **Listing 2**.

To make writing to the I²C interface simple, let's create a write method (see **Listing 3**). In a real code example, you need to handle exceptions.

Now, we need to write 0x00 to the configuration register (0xAC) and send an 0xAA command to start a measurement.

Doing this is quite simple. When you are sending one byte to the DS1621 sensor, consider it to be a command. When you are sending two bytes to the DS1621 sensor, use the first byte as a register number and write the second byte to the register, as shown in **Listing 4**.

After the commands in **Listing 4** are sent, the sensor starts taking continuous temperature measurements; it also starts recording the measurements in an internal register.

We can read a measurement from a ByteBuffer object using the following code:

```
i2c.read(0xAA, 0x1, buffer);
```

Listings 5a and **5b** are a working example. You just need to get an instance of the Thermometer class from the MIDlet startUp() method and invoke the test() method.

Using this code, with the help of Oracle Java ME Embedded and the Freescale FRDM-K64F board, you can always be aware of the temperature in your office or home.

Conclusion

With a new release, Java ME 8 now fully lives up to its design promise of delivering a feature-rich Java 8 platform that scales from powerful embedded systems all the way down to resource-constrained single-chip microcontrollers.

Developers can now rely on a consistent, standards-based programming model and platform that allows true code reuse from large to small solutions. In most cases, the same unmodified application binary will run across the entire range of target devices—regardless of the underlying hardware and software differences. This means faster time to market, improved security and flexibility, and the ability to deliver more product value, faster. `</article>`

LEARN MORE

- Oracle Java ME Embedded documentation portal
- Oracle Java ME Embedded blog
- Terrence Barr's blog
- Embedded Java on Twitter
- Oracle Java ME Embedded forum

LISTING 1 | LISTING 2 | LISTING 3 | LISTING 4 | LISTING 5a | LISTING 5b

```
cfg = new I2CDeviceConfig(0, 0x4f, 7, 100000);
```

Download all listings in this issue as text