Java magazine

By and for the Java community

//SEPTEMBER/OCTOBER 2012 /

JAVA AT SEA

JAMES GOSLING explores the oceans with Liquid Robotics' Wave Glider 17

23 DUKE'S CHOICE AWARDS AT 10

39 BRIAN GOETZ ON PROJECT LAMBDA

ORACLE.COM/JAVAMAGAZINE

ORACLE

//table of contents /

PHOTOGRAPH BY BOB ADLER

Innovation is the lifeblood of our industry—so said Java Community Process Executive Committee member Ben Evans in our interview. And often where there is innovation, there is Java. In this issue, we celebrate the innovative use of Java by profiling the winners of the 2012 Duke's Choice Awards. For this 10th year of the awards, the judging criteria themselves got a little innovative: while the majority of winners were recognized for their technological innovations using Java, two Java user groups (JUGs) were recognized for their innovative approach to bettering the Java community. By acknowledging these JUGs, the awards demonstrate that Java remains a user-driven, not vendor-driven, phenomenon. Meet all of our winners.

Speaking of innovators, our cover story features another Duke's Choice Award winner, Liquid Robotics. As chief software architect, Java pioneer James Gosling is rewriting (in Java) the onboard software for the Wave Glider, a marine robot used for gathering ocean data. While visiting his longtime friend Bill Vass at Liquid Robotics, Gosling thought the technology was so cool that he asked him for a job. Learn more in "Java at Sea."

Java SE 8 is on its way and full of new innovations, including lambda expressions. Read our interview with Java Language Architect Brian Goetz, "Looking Ahead to Project Lambda," to find out how lambda expressions will change the way you code.

Innovation also brings change. While Justin Kestelyn, our founding editor in chief, has moved on to other community-building opportunities, we are committed to making *Java Magazine* better with every issue. I am honored to take the reigns from Justin and am thankful for his countless contributions.

I hope to see many of you at JavaOne (where you'll definitely find innovation).

Caroline Kvitka, Editor in Chief BIO

//send us your feedback /

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

PHOTOGRAPH BY BOB ADLER

MAKE THE FUTURE JAVA

Java™

FIND YOUR JUG HERE

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

Régina ten Bruggencate
JDuchess

LEARN MORE

ORACLE®

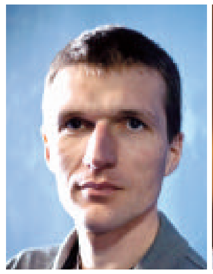
The University of Kent's Michael Kölling (left) describes Greenfoot programming; future developers get hands-on experience.

MAKE THE FUTURE JAVA

More than 300 San Francisco Bay Area students participated in "Make the Future Java," a two-day interactive workshop August 7–8 created to inspire students' love of technology and computer science and to help spark the next generation of Java developers. Professors from Carnegie Mellon University and the University of Kent led hands-on sessions using Alice (visual 3-D educational software for creating animations) and Greenfoot (visual 2-D educational software for creating games and simulations). The Oracle Education Foundation recently awarded grants to Carnegie Mellon's Alice program and the University of Kent's Greenfoot initiative to help awaken student interest and promote a love of computer science, technology, and Java in future technologists and innovators. "As the steward of Java, Oracle is committed to the investment in Java technology and in creative ways to educate a new generation about Java and the opportunities it brings," says Alison Debenwick-Miller, vice president of Oracle Academy. "Our next generation of future technologists and innovators have been given a great start." Learn more about Oracle's Make the Future Java initiative.

PHOTOGRAPHS BY SAUL LEWIS

JAX Innovation Awards 2012



The 2012 JAX Innovation Award winners were announced on July 10 at the JAX Conference in San Francisco, California. The awards celebrate the culture of innovation in the Java ecosystem. The winners are selected in a three-step process: the developer community nominates candidates, an independent expert panel selects five finalists in each award category, and the developer community votes.

The 2012 winners received a US$2,500 prize. This year's winners were Restructure101 (Most Innovative Java Technology); JetBrains (Most Innovative Java Company); **Adam Bien** (Top Java Ambassador, above left); and **Charles Nutter** (Special Jury Award, above right).

PHOTOGRAPHS BY PIOTR MALECKI/GETTY IMAGES, MARKUS EISELE

TDC: THE DEVELOPER'S CONFERENCE

TDC track owners and conference collaborators on stage on the final day (top); Oracle University's Debora Palermo talks about training programs.

More than 3,400 developers attended the Developer's Conference (TDC) in São Paulo, Brazil, in July. This five-day event, organized by Globalcode, is considered the premier multicommunity developer conference in Brazil. "We like to say *multicommunity* rather than *multitechnology* because it is interesting and beneficial when various communities get together," says **Yara Senger**, one of the organizers. "Each community has its own personality, and they learn a lot from each other."

TDC featured more than 250 speakers across 37 tracks. New this year at TDC was the Java University track, sponsored by Oracle. Aimed at university students and professionals new to Java, this track included introductory-level lectures that had an educational focus, as well as practical exercises. Members of Brazilian Java user groups including GoJava, JavaBahia, JavaNoroeste, and SouJava lent their expertise to the Java, SOA, Python, Ruby, mobile, and digital TV tracks to make this a successful and educational event. In the mobile track, developers—including a 15-year-old student—who created a Java ME app received a Nokia phone.

At the TDC opening keynote, **Debora Palermo**, Oracle University country manager for Brazil, discussed Oracle's Workforce Development Program (WDP), which allows educational institutions to deliver Oracle training. WDP enables easy and low-cost access to Oracle training in local communities around the world. "Oracle University is committed to creating the next generation of Java developers, and WDP can make that happen," Palermo said. Oracle University is partnering with Globalcode to offer WDP. Students can earn official Oracle Course Certifications through WDP.

PHOTOGRAPHS COURTESY OF GLOBALCODE

Open Data, Government, and Transparency

The open data movement is growing worldwide. One initiative, the Open Government Partnership, is working to make it easier for the public to find and access government data from around the world. At the Developer's Conference (TDC) in São Paulo, Brazil, a new track called Open Data focused on what a developer needs to know to build applications using open data. Track leaders presented successful open data projects and discussed the Semantic Web, big data sets, data visualization, and designing APIs.

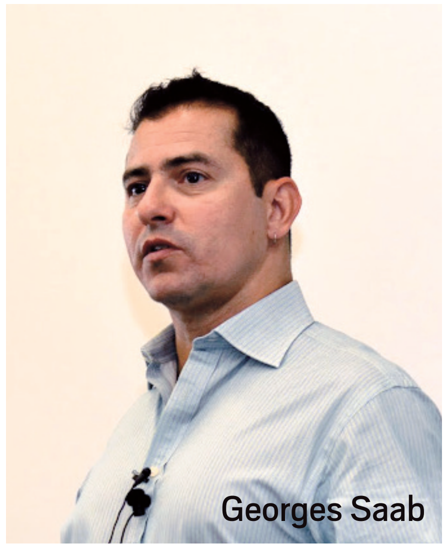
The track finished with a full Transparency Hacker Day devoted to creating applications using existing open government data, including the Brazilian government's release of most officials' salaries. Participants created an application called Serve o Quanto Ganha (Worth What You Earn), which looked at two public employees and compared how many salaries of *X* are needed to pay *Y*'s salary.

Learn more about open data at the U.S. government open data project, the Brazilian government open data project, the U.K. government open data project, and the 2012 International Open Government Data Conference.

Rich Hickey

Rémi Forax

John Rose


Georges Saab

2012 **JVM LANGUAGE** SUMMIT

The 2012 JVM Language Summit, held at Oracle's Santa Clara, California, campus July 30–August 1, brought together the world's leading Java Virtual Machine (JVM) language architects. As one attendee tweeted, "If the building collapses, progress in language innovation will be set back by 100 years." The air was crackling with code.

Speakers included **Georges Saab** (introduction), **William Cook** (batches), **Mark Roos** (RTalk), **Matt Fowles** (staged compilation), **Jochen Theodorou** (Groovy, invokedynamic), Oracle's **John Rose** (arrays) and **Brian Goetz** (Project Lambda), **Dan Heidinga** (MH introspection), **Lukas Stadlar** (Truffle), **Per Bothner** (Kawa), **Jeroen Frijters** (CLR/JVM), **Duncan MacGregor** (migrating to JVM, ASM), **Rich Hickey** (Datomic), **David Chase** (Fortress), **Rémi Forax** (JDart), **Basil Hosmer** (Mesh), **Ryan Sciampacone** (multitenant JVM), **Jim Laskey** (JDI), **Doug Simon** (Graal), **Michael Wiedeking** (AL1 JVM Assembler), **Andrey Breslav** (Kotlin/Java interoperability), **Gilles Duboscq** (Graal), and **Donald Raab** (collections).

"It's one of the best conferences I've ever seen," remarked one attendee. "No commercials, no sales, just a communal focus on technology in a spirit of open learning and curiosity. Here are the people who invent programming languages." Another wore a T-shirt that said, "Changing lives one line at a time."

Videos of presentations are available online.

PHOTOGRAPHS BY OLEG PLISS

The JCP: Open and Transparent

The JCP.Next JSRs mandate new openness and transparency for the Java Community Process (JCP). For example, JSR 348 requires that the JCP invite the global public to two Executive Committee (EC) meetings each year. The first of these was held on June 26. Learn more in the meeting minutes.

The JCP will be busy at JavaOne. It will host four sessions: JCP.Next: Reinvigorating Java Standards; 101 Ways to Improve Java: Why Developer Participation Matters; Meet the JCP Executive Committee Candidates; and The JCP and OpenJDK: Using the JUGs' Adopt Programs in Your Group. The JCP will also hold three events: JCP EC Face to Face Meeting; JSR Expert Group Spec Lead Gathering; and the JCP party on October 2, where the 10th JCP Annual Awards will be presented. This year's award categories are JCP Member/Participant of the Year, Outstanding Spec Lead, and Most Significant JSR.

FEATURED JAVA.NET PROJECT

VISUALLANGLAB

VisualLangLab is the world's first fully visual, code/script-free parser-generator integrated development environment (IDE). It depicts grammar rules as trees containing nodes with intuitive icons. These "rule trees" can be run instantly at the click of a button without code generation or any other intermediate step. The visual representation enables even users without special skills to utilize VisualLangLab to create sophisticated parsers. These parsers can be embedded into client programs written in any Java Virtual Machine language.

VisualLangLab was highlighted in Java Posse Episode 364. The Posse noted, "If you are observing the crop of new languages springing up at present and think to yourself, 'I could do better than that,' well, now's your chance. VisualLangLab is a graphical tool for designing language grammar. And it's not just for full-blown languages. The project could be used for developing free-form domain-specific languages."

The VisualLangLab project was started in January 2011, but its roots go back to 2004 when founder **Sanjay Dasgupta** was leading a team in developing a tool chain for a custom business-oriented language. As the work progressed, Dasgupta realized that the classic parser-generators (ANTLR, JavaCC) were too complex for many users. So Dasgupta began working on VisualLangLab's predecessor.

To date, more than 1,400 developers have downloaded VisualLangLab. Through all the years of development effort, Dasgupta has been the project's only developer. He cites the complexity of the code as being a likely deterrent to other developers. However, he is trying to ease the path for people who might like to join the project by supplementing the code's internal documentation.

Learn more in "VisualLangLab—Grammar Without Tears" and visit the project on Java.net for a tutorial, FAQs, and more.

JAVA CHAMPION PROFILE

KIRK PEPPERDINE

Java Magazine: Where did you grow up?

Pepperdine: I was born in Ottawa, Ontario, Canada; started school in Clinton, Ontario; moved to Saint John, New Brunswick; and then back to Ottawa.

Java Magazine: How did you first become interested in programming?

Pepperdine: My first year of biochemistry ended in an economic downturn. I had no summer job, so I enrolled in a programming course. It just clicked. I started using computers to help me do my biochem labs.

Java Magazine: What was your first programming language?

Pepperdine: The first language was PL/C on an IBM 370, but I quickly switched to using Fortran for schoolwork. My first professional language

was BASIC.

Java Magazine: What was your first programming job?

Pepperdine: When I started working as a biochemist, I had to do a lot of programming. In one memorable experiment, we wrote software to control a pressure relief valve for a high-frequency oscillator ventilator circuit. We had to burn the program into a ROM on a chip that ran Tiny BASIC. Input came from another Intel chip that created a digital signal from air pressure. The two chips and a solenoid were all wire-wrapped onto a board and then put into the airway circuitry. An airway is warm, very moist, with salty air. The board would work for about four hours, after which I had to wash, dry, and then autoclave it. That was a lot of fun.

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

Pepperdine: Continuing my efforts to change peoples' ideas on how to tune applications. I've recently started a performance tooling company called jClarity. We are building some experimental software that doesn't look like anything else out there.

Read more about Kirk Pepperdine and visit Kodewerk for information on his Java performance tuning courses.

MoroccoJUG members filled the room at a recent meeting.

JAVA USER GROUP PROFILE

MoroccoJUG

MoroccoJUG—the only Java user group (JUG) in Morocco—was founded in early 2009 by **Badr Elhouari** (technical expert at Safran Morpho) and **Faissal Boutaounte** (technical leader at Strategum SARL). Although the JUG is located in Rabat, its activities extend to Casablanca, Marrakech, Mohammadia, Tangier, and other cities.

MoroccoJUG has three main objectives: discussing Java and strengthening the Java community in Morocco; helping universities and engineering schools educate students about Java and related technologies; and helping to move Java forward in Morocco and around the world.

MoroccoJUG also seeks to organize and grow the Moroccan developer community and enhance its global recognition through informative monthly meetings and the Java Education Moroccan Initiative (JEMI). JEMI provides Java courses to students in their graduation year.

The courses have led to higher attendance at MoroccoJUG meetings, with membership growing to 700 people in less than three years.

Another major focus for the MoroccoJUG leaders is global outreach. MoroccoJUG was one of the first JUGs affiliated with the JUG-Africa umbrella group, contributing to the JCertif effort from its inception. The JUG is active in the Adopt-a-JSR program and is one of three JUGs that volunteered to work in the AdoptOpenJDK Bugathon Instructors Training pilot initiative.

MoroccoJUG actively participates in major Java conferences such as Devoxx, Jfokus, Java Developer Conference, and the IOUC Summit. The JUG is organizing an international Java event called JMaghreb that will be one of the biggest Java technology events in North Africa.

Learn more on Java.net or Twitter (@moroccojug).

PHOTOGRAPHS COURTESY OF MOROCCOJUG

ECLIPSE JUNO RELEASED

The Eclipse Community released its latest version, code-named Juno, in late June. This continued a nine-year tradition of annual releases of the Eclipse suite in June. Seventy-one projects, including nine new projects, make up the Juno release.

At Juno's core is Eclipse version 4.2. Important new features include Code Recommenders, which significantly smartens Eclipse code completion; a new Nano kernel for the Virgo Web server, which facilitates building very small OSGi-based applications; and added support in Xtext for integrated debugging of domain-specific languages created using Xtext.

The Eclipse Juno represents the work of 445 open source committers on 55 million lines of code, and the participation of more than 40 Eclipse member companies.

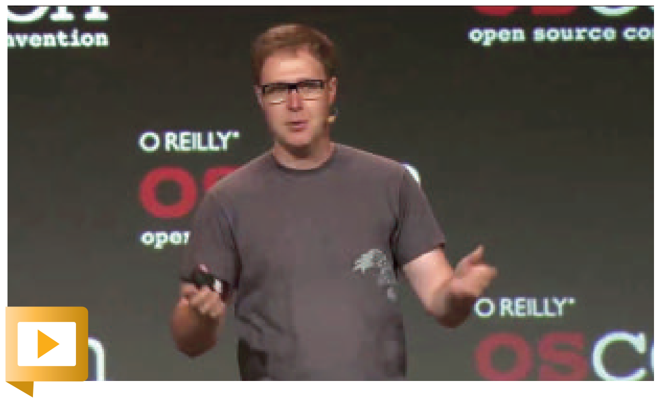
LAMBDA EXPRESSION:

Music to Program By

In response to last issue's call for Java songs, **Freddy Guime** told us about his one-man band, Lambda Expression, and pointed us to its current song catalog. Guime's latest (and his favorite) song is the edgy "In the Zone," where he sings about those wonderful (if rare) programming days where everything just flows, the code is seeming to almost write itself, and the last thing you want is to be interrupted by meetings, e-mail, and life. Guime also blogs and posts podcasts at Java Pub House. Do you have a Java song? Share it with us!

OSCON Offers Hard and Soft Skills

OSCON, the open source convention, was held in Portland, Oregon, in July. With more than 3,600 attendees and more than 4,600 virtual attendees watching the live stream, OSCON provided great information about open source technologies, communities, and strategies. Java was well represented with the sessions High Performance Network Programming on the JVM; Hacking JavaFX with Groovy, Clojure, Scala, and Visage; The Art of Metaprogramming in Java; and The Java EE 7 Platform: Developing for the Cloud. OSCON included a focus on "soft skills," with tracks for Community and Geek Lifestyle. Sessions such as Open Source 2.0: The Science of Community Management and Analyzing How Developers Learn Online for Fun and Profit reminded developers that coding chops can get you into an open source community, but how you interact with others has a big impact on your long-term success. Speaker slides and videos are available at oscon.com.



David Eaves on the science of open source community management

FIVE GREAT JAVA BLOGS

There are a lot of excellent Java and Java Virtual Machine (JVM) technology blogs out there. But if you want to stay informed on current news and what's coming up in the future, subscribing to these five blogs is a great way to start:

1 Adam Bien's Weblog: **Adam Bien**—Java Champion, author, and consultant—covers almost all things Java in his blog, with a particular focus on Java EE.

2 FX Experience (JavaFX News, Demos and Insight): **Jonathan Giles** provides JavaFX news, interviews, links of the week, release announcements, and programming tips.

3 Geertjan's Blog (Random NetBeans Stuff): **Geertjan Wielenga**, Oracle Developer Tools principal product manager, says that he focuses his posts on "NetBeans, with an occasional reference to NetBeans, and sometimes diverging to topics relating to NetBeans. And then there are days when NetBeans is mentioned. . . ."

4 Inspired by Actual Events (Dustin's Software Development Cogitations and Speculations): Developer **Dustin Marx** shares his observations on a wide range of topics including Java, the JVM, JavaFX, and Groovy.

5 There's not a moment to lose!: Oracle Java Platform Chief Architect **Mark Reinhold** doesn't post frequently, but when he does, it's often significant news.

ART BY I-HUA CHEN

PHOTOGRAPH BY ELIJAH MERCHÁN

EVENTS

Java Embedded @ JavaOne *OCTOBER 3–4, SAN FRANCISCO, CALIFORNIA*

At Java Embedded @ JavaOne, C-level executives, architects, business leaders, and decision-makers from around the world come together to learn how Java Embedded technologies and solutions offer compelling value and a clear path forward to business efficiency and agility. This new program at JavaOne features dedicated business-focused content that delves into how Java Embedded delivers a secure, optimized environment ideal for multiple network-based devices, while industry-focused sessions show how Java Embedded technologies are being successfully utilized.

OCTOBER

Silicon Valley Code Camp
OCTOBER 6–7
LOS ALTOS HILLS, CALIFORNIA
At this community event, developers learn from each other. Featured topics include software branding, legal issues around software, and much more. All are welcome to attend and speak. More than 2,000 developers attended in 2011.

TDC2012 Goiânia
OCTOBER 20–21
GOIÂNIA, BRAZIL
The Developer's Conference, or TDC, is a multicommunity developer conference with locations throughout Brazil. Tracks include Java, SOA, Mobile, and more.

QCon Hangzhou
OCTOBER 25–27
HANGZHOU, CHINA
This enterprise software development conference is designed for team leads, architects, and project managers and is organized by and for the community. Topics include architecture, functional languages, mobile, and agile engineering practices.

NOVEMBER

W-JAX
NOVEMBER 5–9
MUNICH, GERMANY
W-JAX: The Conference for Java, Web, Architecture, Agile, and Cloud, presents a holistic approach to Java technologies. Topics include software architecture, agile management methods, and issues of enterprise architecture.

QCon San Francisco
NOVEMBER 7–9
SAN FRANCISCO, CALIFORNIA
This practitioner-driven conference is designed for team leads, architects, and project managers. The program includes two tutorial days and three conference days with 18 tracks on a wide variety of software development topics.

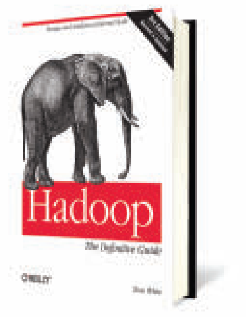
Devoxx
NOVEMBER 12–16
ANTWERP, BELGIUM
This annual European Java conference is organized by the Belgian Java User Group. It features a two-day in-depth Devoxx University program, followed by three days of conference sessions. Hands-on labs and Birds-of-a-Feather sessions are on the agenda.

COMING SOON: JDAYS

A new Java conference, jDays, will take place December 3–5 in Gothenburg, Sweden. The first two days of the conference feature a lineup of sessions, with a unique twist: the community can vote on which session abstracts they find most interesting, and see the current results on the jDays home page.

Day 3 of the conference also offers something different: a full day of free courses and hands-on training. Conference co-organizer **Hamid Samadi** (above), who founded the Gothenburg Javaforum user group in 2005, will leverage his experience as an entrepreneur and trainer by actively participating in the jDays Day 3 activities.

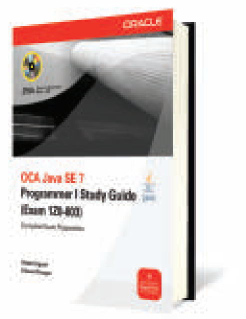
JAVA BOOKS



[*HADOOP: THE DEFINITIVE GUIDE, THIRD EDITION*]

By Tom White
O'Reilly Media (May 2012)

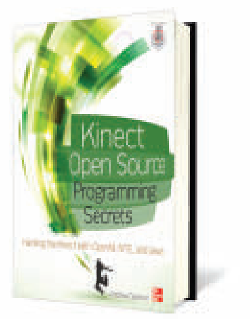
Ready to unlock the power of your data? With this comprehensive guide, you'll learn how to build and maintain reliable, scalable distributed systems with Apache Hadoop. This book is ideal for programmers looking to analyze data sets of any size, and for administrators who want to set up and run Hadoop clusters. It includes case studies that demonstrate how Hadoop is used to solve specific problems. This third edition covers recent changes to Hadoop, including material on the new MapReduce API, as well as MapReduce 2 and its more flexible execution model (YARN).



[*OCA JAVA SE 7 PROGRAMMER I STUDY GUIDE (EXAM 1Z0-803)*]

By Robert Liguori and Edward Finegan
Oracle Press (October 2012)

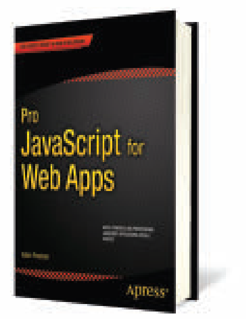
This book offers complete coverage of the updated Java SE entry-level programmer certification exam and provides in-depth, up-to-date coverage of all the exam objectives. It offers an integrated study system based on proven pedagogy—step-by-step exercises, Exam Watch, Inside the Exam, and On-the-Job notes. In addition, chapter self-tests help reinforce and teach practical skills while preparing you for the exam. The included MasterExam practice exam software features more than a hundred questions.



[*KINECT OPEN SOURCE PROGRAMMING SECRETS: HACKING THE KINECT WITH OPENNI, NITE, AND JAVA*]

By Andrew Davison
McGraw Hill Professional (April 2012)

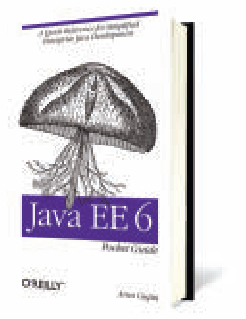
The Kinect motion-sensing device for the Xbox 360 and Windows became the world's fastest-selling consumer electronics device when it was released (8 million sold in its first 60 days) and won the T3 Gadget Awards 2011 Gaming Gadget of the Year. Learn how to harness the Kinect's powerful sensing capabilities for gaming, science, multimedia projects, and more on platforms running Windows, Mac OS, and Linux.



[*PRO JAVASCRIPT FOR WEB APPS*]

By Adam Freeman
Apress (June 2012)

This book gives you all of the information you need to create professional, optimized, and efficient JavaScript applications that will run across all devices. It takes you through all aspects of modern JavaScript application creation, showing you how to combine JavaScript with the new features of HTML5 and CSS3 to make the most of the new Web technologies. The focus of the book is on creating professional Web applications and ensuring that your app provides the best experience for your users, with smooth and responsive control and feedback.



[*JAVA EE 6 POCKET GUIDE*]

By Arun Gupta
O'Reilly Media (September 2012)

This guide provides an overview of the main technologies in the Java EE 6 platform, including extensive easy-to-understand code samples that demonstrate many improvements. Whether you're familiar with Java EE 5 or a Java programmer approaching the enterprise edition for the first time, this book will quickly get you up to speed on Java EE 6. You'll discover how Java EE 6 provides a simplified developer experience and improves on the developer productivity features introduced in Java EE 5. You'll also delve into Java EE 6 profiles.

JCP Executive Series

A Conversation with Ben Evans

London JUG representative **Ben Evans** discusses the JCP and the Java community. **BY JANICE J. HEISS**

PHOTOGRAPHY BY JOHN BLYTHE

In the second of a series of interviews with distinguished members of the Executive Committee of the Java Community Process (JCP), we sat down with London, England, Java user group (JUG) representative Ben Evans, who has been a professional developer and open source enthusiast since the late '90s. Evans was the lead performance testing engineer for the Google IPO (the largest auction ever conducted), worked on the initial U.K. trials of 3G networks with BT, built award-winning Websites for some of Hollywood's biggest hits of the '90s, rearchitected and reimagined technology that helps some of the most vulnerable people in the U.K., and has worked on everything from some of the U.K.'s very first e-commerce sites to multibillion-dollar foreign exchange trading systems. He is currently CEO of jClarity, a London startup working on automation of Java performance tuning.

Evans is the coauthor, with Martijn Verburg, of The Well-Grounded Java Developer, *which focuses on Java SE 7. He currently represents London's JUG—the London Java Community (LJC)—on the Executive Committee of the JCP.*

Ben Evans shared the stage with Oracle's Brian Goetz (left) and Mark Reinhold at the Devoxx 2011 Java developer conference in Antwerp, Belgium.

LJC Wins a Duke's Choice Award

The London Java Community (LJC) is the winner of a 2012 Duke's Choice Award, which celebrates "extreme innovation in the world of Java Technology" and is granted to the most-innovative projects that use the Java platform. The LJC was recognized for its work in advancing the Java ecosystem, including growing a large and vibrant JUG community, being the first JUG to have a seat on the JCP Executive Committee, and spearheading the Adopt-a-JSR and Adopt OpenJDK programs.

Java Magazine: Tell us about your involvement with Java, the London JUG, and the JCP.
Evans: I've been programming in Java since 1998 and am currently CEO of jClarity, a London startup focusing on automation of Java performance tuning. I help run the London JUG, which was successfully elected to the Executive Committee of the JCP last year, and I've been involved with the JCP for nearly 18 months.

I personally got involved with the JCP through involvement in the user group, where I met Martijn Verburg. We started writing a book together, and I started doing more public speaking and became more involved in the community generally. My involvement in the JCP developed out of that. I personally wanted our JUG to become more of a community participant. In my day job, I had moved from a position where Java was just one language among several that I used to a position where Java was my primary language, and I wanted to participate and help unlock more of the platform's potential.

When we originally stood for election to the Executive Committee we quite reasonably thought, "There's no way we're going to be elected, because we're up against some really established players." We thought we might just stir things up a bit and perhaps stimulate some debate—but we honestly didn't think we'd win. So it was quite a surprise to us that we actually won—something of a happy accident.
Java Magazine: How do you address the problem of the relationship between a JUG and the larger Java community?
Evans: With Java, we have what is known as an iceberg problem. So even within a relatively engaged JUG, the percentage of people who aren't part of the JUG far exceeds the number who are. We have roughly 2,500 JUG members in London, and we estimate that we're reaching 3 to 5 percent of the Java developers in the area, which is incredibly small.

So there is a vast iceberg of Java developers under the surface. I'm not

The JCP's Ben Evans believes that Java suffers from an "iceberg problem," exemplified by the small percentage of London Java developers who are part of the London JUG.

ON DISRUPTION
We thought we might just stir things up a bit and stimulate some debate—but we honestly didn't think we'd win.

entirely sure what to do about that, but it's a reality. We discovered somewhere around the summer and fall of 2011 that there are plenty of multitalented people who were interested in specific JSRs and emerging Java technologies.

Java Magazine: Who shows up at the London JUG meetings?

Evans: It's quite varied, with a lot of business representation in financial services, insurance companies, and telcos, plus a lot of academics with a growing number of undergraduates. More recently, we have people from the startup community that is taking hold in East London.

Java Magazine: How can we increase participation in the JCP?

Evans: We need to keep reminding people that we're here, and that the decisions and new standards that come out of the JCP will affect the future careers of every Java developer, and that the process and developing standards need their help. There are a number of ways to join—as an individual, a company, or even a user group. And there are a number of ways to participate as a member. You can follow the progress of standards that you're particularly interested in and ensure that existing standards and OSS [open source software] projects are represented in JSRs that touch on any patents' prior art. You can blog about new standards and help raise awareness. I'm sure that better outreach, better communication, and trying to get more everyday developers involved will all lead to better standards.

Too many Java developers have still never heard of the JCP or think that it went away after the Oracle acquisition.

Java Magazine: In walking the fine line between respecting standards and encouraging innovation, does the JCP err too far in one direction?

Evans: Walking this tightrope is really difficult. Innovation is the lifeblood of our industry, but we need strong compatibility requirements. Without them, we impose costs and externalities on developers and end users of the platform. Too much of that, and people will start leaving. I think the JCP mostly gets it right, but we still need to keep a close eye on those factors.

Java Magazine: What is the best thing that has happened to the JCP recently?

Evans: There's been a resurgence of interest in the JCP in the last year. There's lots of new blood, a new version of the process that specifically makes transparency a key principle, and new momentum for change—it's tough to pick just one thing.

Java Magazine: Would you like to see any structural changes in the JCP?

Evans: I think the moves to merge the two committees, and to revisit the most complex section of the JSPA [Java Specification Participation

Increasing Openness and Transparency at the Java Community Process

At the Java Community Process (JCP), three JSRs—two still in the works—are increasing openness and transparency.

JSR 348, also known as JCP .next.1, was completed in October 2011 and focused on some small and simple but important changes that enhance transparency and enable broader participation. "We're already seeing the benefits of these changes as new and existing JSRs adopt the new requirements," remarks JCP Chair Patrick Curran.

JSR 355, JCP Executive Committee Merge, which simply merges the two executive committees into one for greater efficiency and to encourage synergies between the Java ME and Java SE platforms, will be completed later in 2012.

JSR 358, also known as JCP.next.3, filed in June 2012, constitutes a major revision of the JCP. It will modify the Java Specification Participation Agreement (JSPA), the legal agreement that members sign when they join the organization, as well as the Process Document, and it will tackle a large number of complex issues, many of them postponed from JSR 348. Due to its complexity, JSR 358 should be completed toward the end of 2013 or possibly later.

Topics to be addressed in JSR 358 include the role of independent implementations (those not derived from the Reference Implementation), licensing and open source, ensuring that new transparency requirements are implemented correctly, compatibility policy and Technology Compatibility Kits (TCKs), the role of individual members, patent policy, and IP flow.

All JCP business is being conducted in the open and can be followed and participated in on java.net.

Agreement]—namely, the parts that deal with intellectual property and how it is handled in the JCP—are great ideas. These are the most important areas where change is overdue.

In terms of practical changes, the JCP lacks a proper calendaring system to remind Executive Committee members when votes are in-flight. We certainly need to fix that.

Java Magazine: Are you happy with the Expert Group processes in the JCP? What changes would you like to see take place?

Evans: The trick here is to balance the autonomy of spec leads with the desire to have standard processes. My personal thoughts are that having standardized procedures—licenses for reference implementations, TCKs [Technology Compatibility Kits], and so on—can help, and that spec leads should be encouraged, but not mandated, to use them.

The JCP should be collaborating with the community, with the community providing raw material and feedback, so we can see what's catching on and where we need standards. A good example of this would be the JSON standards. We're in need of a decent JSON library to compete with other trendy languages that are perceived as having better growth.

To help this collaboration, the LJC launched the Adopt-a-JSR and Adopt OpenJDK programs to directly connect ordinary developers with the groups working on new standards and new language technology. This ensures that new standards and features get early feedback from the end users who will ultimately end up using them in their day jobs for years to come.

We can't change the language willy-nilly, because it affects millions of developers and huge numbers of people in industry and business worldwide. So as keepers of the standards, we have to go slowly and carefully. We can't make changes at the internet speed of a language like Scala. On the other hand, new languages in those ecosystems can serve as a kind of language lab—we can find out what works and what doesn't. There is an upside to being forced to go slowly.

Java Magazine: Finally, has Oracle delivered on the promise of increased transparency and openness in the JCP?

Evans: All the indications I've seen have been positive. I have no complaints. </article>

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

LEARN MORE

- Java Community Process
- Ben Evans' London Java Community blog

Data Quality Tools for Java

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

BEFORE

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

AFTER

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

*Highlights indicate added and/or corrected data.

Realtime NCOA Link Change-of-Address Web Service available

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

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

Multiplatform

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

MELISSA DATA®
Your Partner in Data Quality

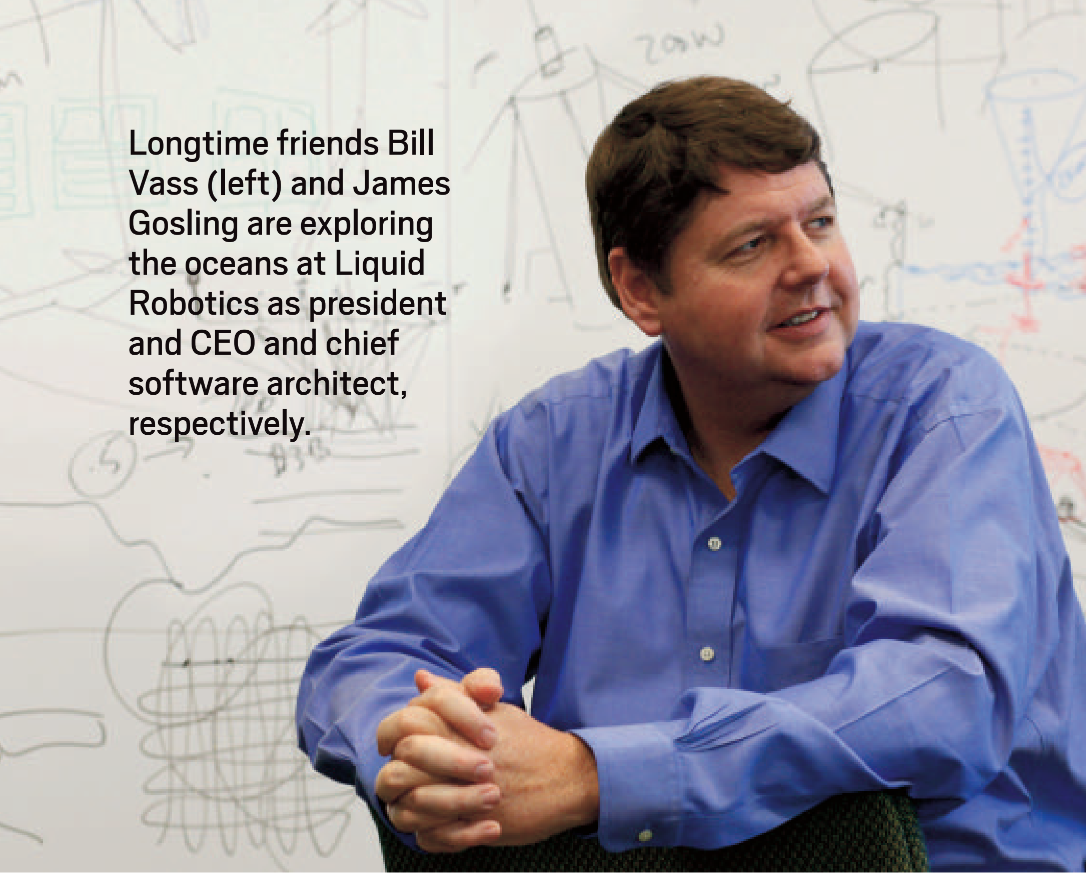
JAVA AT SEA

Liquid Robotics charts a new course with help from Java pioneer James Gosling.

BY DAVID BAUM

A Wave Glider ready for deployment in Moss Landing, California

PHOTOGRAPH COURTESY OF LIQUID ROBOTICS


Longtime friends Bill Vass (left) and James Gosling are exploring the oceans at Liquid Robotics as president and CEO and chief software architect, respectively.

SNAPSHOT

LIQUID ROBOTICS, INC.
liquidr.com

Headquarters:
Sunnyvale, California

Industry:
Maritime data collection, observation, monitoring, and exploration

Employees:
90

Java technologies used:
JDK 7 with NetBeans, Swing, and the NASA World Wind Server

PHOTOGRAPHY BY BOB ADLER

Java's success is partially attributable to its immense versatility. From animating internet programs to powering cell phones, it's the most popular technology language in the world. Some of the most interesting Java programs are embedded in scientific research equipment, such as the Wave Glider—the flagship product of Sunnyvale, California–based Liquid Robotics and one of this year's Duke's Choice Award winners (see "Duke's Choice Awards" on page 23).

The Wave Glider is a self-propelled autonomous marine robot that taps the energy of ocean waves for propulsion. Above water, it looks like a high-tech long board, a buoyant platform covered with solar cells and various types of test and measurement equipment. Down below is a submerged glider with wing-shaped panels that have been specially constructed to convert the up-and-down motion of waves into forward thrust. This patented wave motion system permits the craft to travel indefinitely without relying on batteries or fossil fuels, as it only uses renewable energies. No fuel required, no personnel needed, and no harmful environmental emissions.

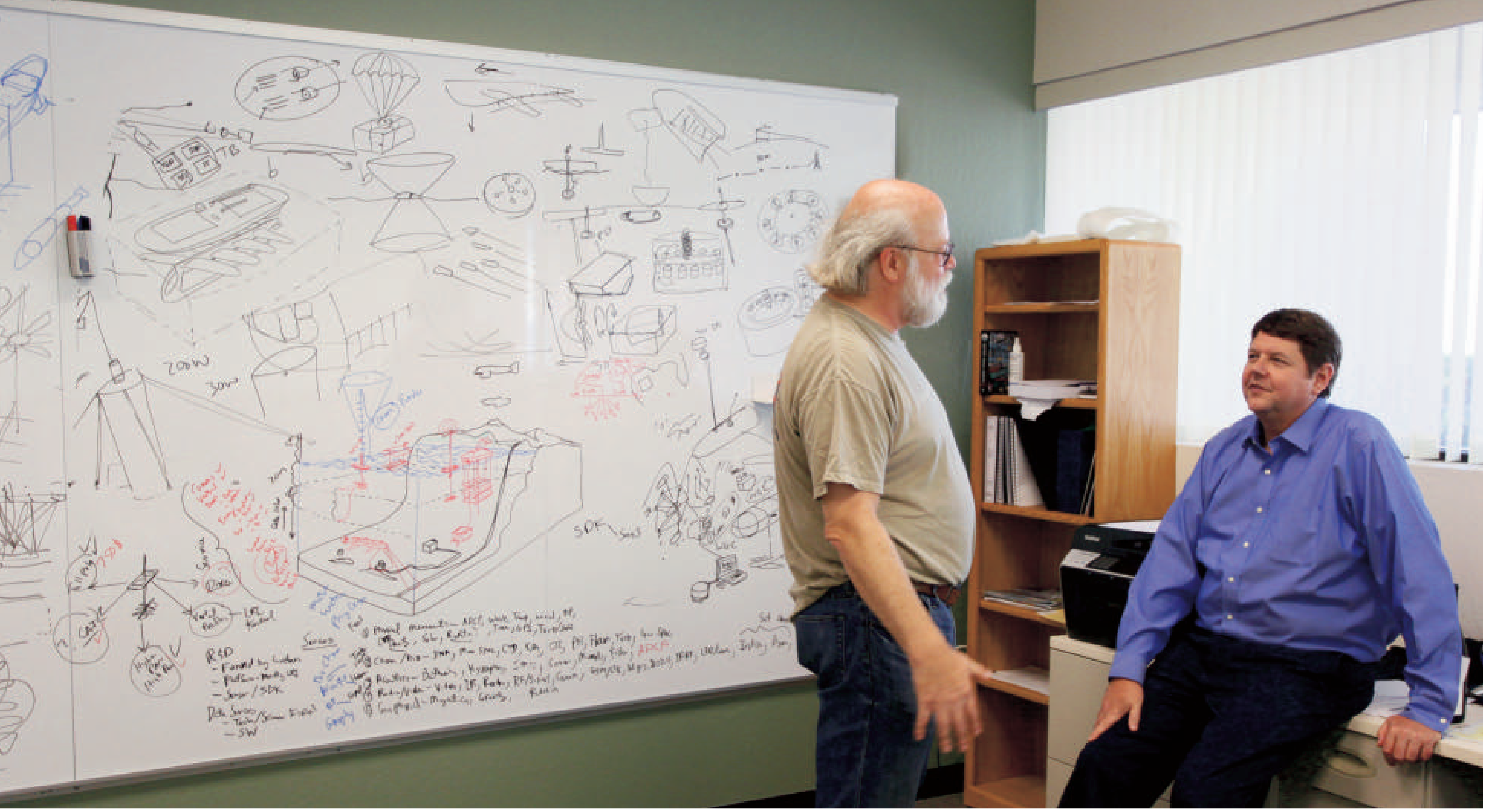
Since James Gosling took over as chief software architect at Liquid Robotics, Java has played a progressively more important role in the missions of these unique oceangoing robots in data transmission and analysis. Best known for his role as a computer scientist at Sun Microsystems, where he invented Java, Gosling is rearchitecting the onboard software and refining a data-as-a-service cloud to provide direct, real-time access to ocean information.

LIFE'S CHANGES

In September 2011, not long after leaving Oracle and a brief stint at Google, Gosling joined Liquid Robotics—a rapidly growing company he had heard about from his longtime friend and coworker Bill Vass, who now serves as president and CEO.

"I visited Bill to see his new venture, and it was just so incredibly cool," Gosling recalls. "They had devised a way to obtain a wide variety of detailed ocean data more cheaply and pervasively than by any other means. The Wave Glider involves an interesting data problem and a large-scale control problem, both of which have been passions of mine for years. So I asked him: 'Do you have a job for me?'"

Vass was quick to hire Gosling and turn him loose on a variety of challenging projects. As chief software



Gosling updates Vass on reworking the Wave Glider's onboard software in Java and developing a data-as-a-service cloud for real-time access to ocean information.

architect, Gosling is responsible for retooling a number of command and control systems and improving network communications, which entails uploading data from each mission to a cloud-based data repository. They are also creating a data visualization system for tracking, monitoring, and controlling Wave Glider missions.

"So much of our platform depends on software," Vass says. "Wave Gliders are like floating data centers on the ocean. They have a lot of onboard intelligence, which is connected via satellite to the cloud."

DATA CENTER AT SEA

"So much of our platform depends on software. Wave Gliders are like floating data centers on the ocean. They have a lot of onboard intelligence, which is connected via satellite to the cloud."

—Bill Vass, President and CEO, Liquid Robotics

MODERN FABLE

Each Wave Glider can collect and transmit oceanic data on a continuous basis on missions that can last as long as a year and cover thousands of miles. To date, collectively Wave Gliders have traveled more than 250,000 nautical miles around the world's oceans. "If you could imagine doing a leisurely stroll across the Pacific Ocean, that's what our robots do," explains Gosling.

Traveling across the ocean on an average of 1 to 2 knots might seem like a tedious undertaking. But if you want to deploy a stationary monitoring system, the alternative is to hire an oceangoing ship to tote it to its destination, at a cost of US$50,000 to US$150,000 per day. Like the fable of the tortoise and the hare, Wave Gliders may be slower, but they are reliable, tireless, and have an inexhaustible source of energy at their disposal. "Slow" turns out to be perfect for scientific instruments: they can carefully collect dense data sets rather than speeding by and missing detail.

From a cost perspective, slow and steady wins every time—a fact that has not escaped the notice of eminent research institutions such as the National Oceanic and Atmospheric Administration, Woods Hole Oceanographic Institution, the Monterey Bay Aquarium Research Institute, Scripps Institution of Oceanography, and the University of Hawaii, all of which now use Wave Gliders for oceanographic experiments.

"Deep-water buoys are remarkably expensive, and the vast majority of that expense is due to getting out into the ocean to do deployment and maintenance," Gosling explains. "If a glider needs maintenance, you send a new one out to replace it and the old one drives itself back to shore."

Monitoring the weather is just one of many uses for the Wave Glider. The autonomous robots are also deployed for commercial operations such as offshore oil and gas exploration, national

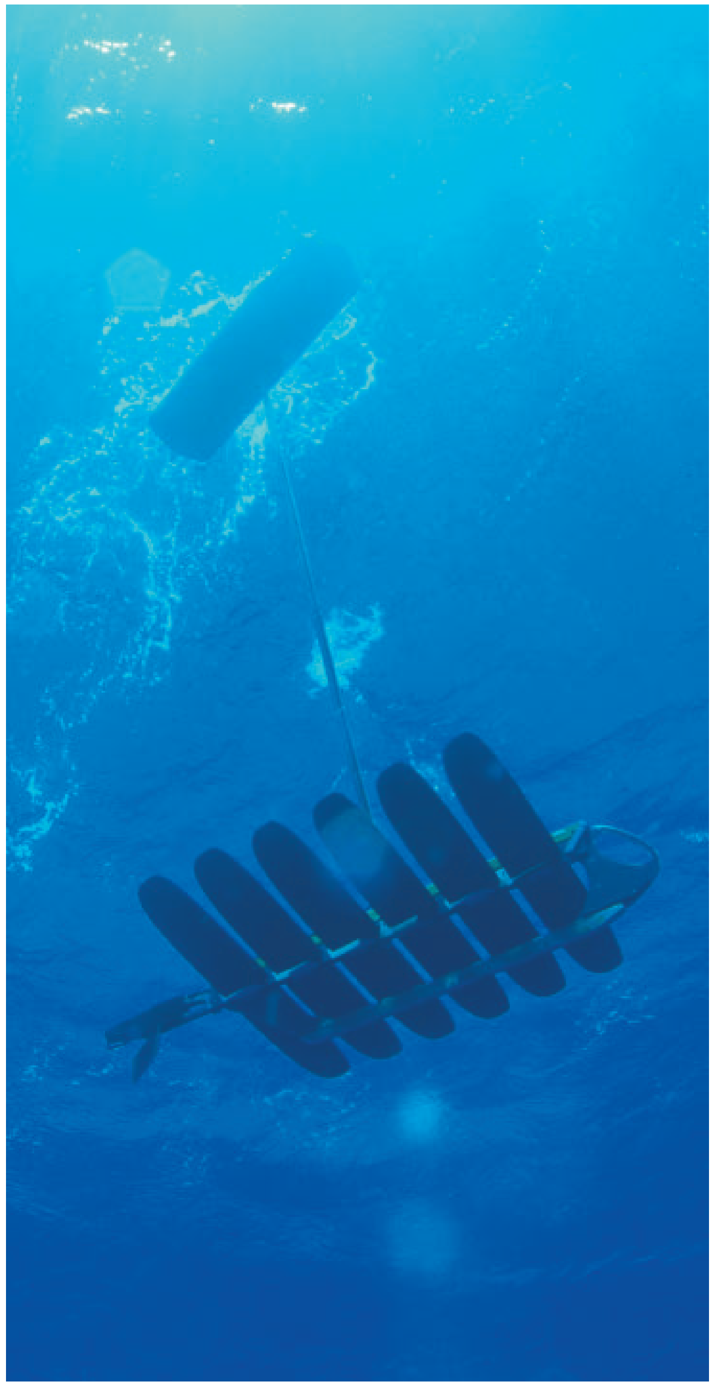
SLOW AND STEADY
"If you could imagine doing a leisurely stroll across the Pacific Ocean, that's what our robots do."
—James Gosling, Chief Software Architect, Liquid Robotics

defense, and environmental operations such as monitoring water temperatures in the Arctic or measuring the carbon levels in the ocean. Information gleaned from onboard instruments helps scientists to collect data about ocean currents, predict hurricanes, detect oil spills, study marine life, and a host of other applications. Solar cells generate electricity for the computers and navigation systems as well as for Iridium modems that link the crafts to shore.

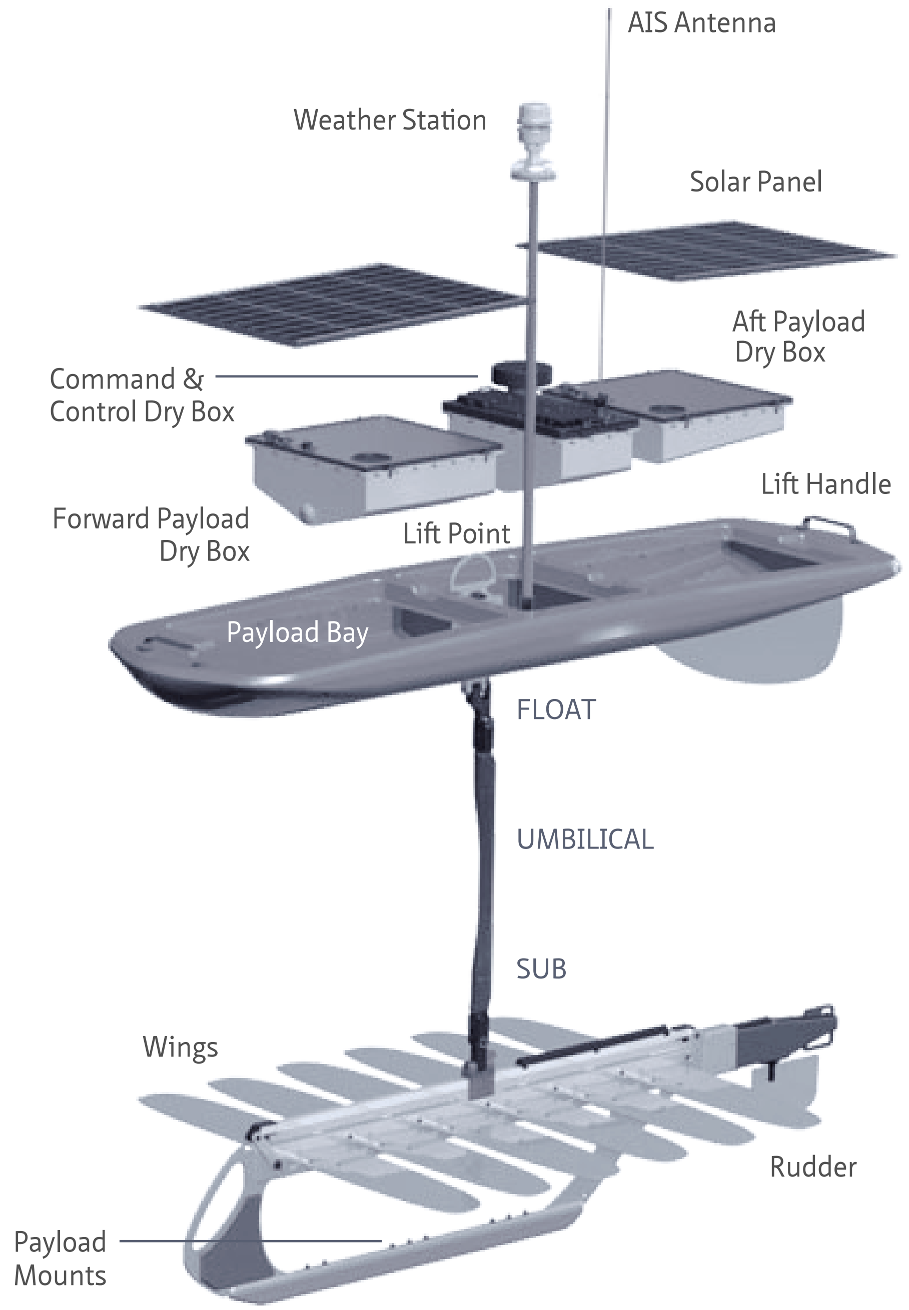
HANDS ON

Gosling likes being back on a small development team where he can have his hand in the code. Most of the existing Wave Glider's onboard software is written in C running on a small microcontroller. In some cases this framework has reached the limits of its complexity, so he and his team are systematically rewriting the command and control components in Java to run on a modern ARM processor.

In addition, Gosling is using Java to develop a data-as-a-service cloud to provide a framework for integrating customer applications. "It's hard to expand these C programs into


PHOTOGRAPH COURTESY OF LIQUID ROBOTICS

Clockwise from top right: a Wave Glider as seen from below; Vass and Gosling at Liquid Robotics' headquarters, where large monitors display Wave Glider activity and data; Vass talks with members of the Liquid Robotics team in the Wave Glider piloting room; Vass and Gosling in the Wave Glider manufacturing area.



This schematic shows the basic configuration for the Wave Glider. Essentially a floating truck, the solar-powered Wave Glider is equipped with systems for navigation, payload control, and satellite communications.

COURTESY OF LIQUID ROBOTICS

sophisticated applications for advanced navigation and swarming," he explains. "Java gives us greater flexibility."

Gosling is also developing desktop visualization software to help customers see what the robots are doing. He uses JDK 7 with components from the Swing GUI widget toolkit and the NASA World Wind mapping component.

Part of the Java Foundation Classes, Swing provides an API for creating flexible GUIs. World Wind is an open source map interface developed by NASA and the open source community for use on personal computers, similar to Google Earth. The program overlays NASA and U.S. Geological Survey satellite imagery, aerial photography, and topographic maps to enable customers to follow their Wave Gliders as they wander through the world's oceans.

"This new interface will include readouts from the various instruments so you can toggle back and forth through time to re-create the history of each mission," Gosling explains. "We are designing it so you can integrate telemetry data from multiple craft to see the tracks of other ships in the neighborhood. If you have to pass through a shipping lane, you can project an animated timeline and correct your course if necessary."

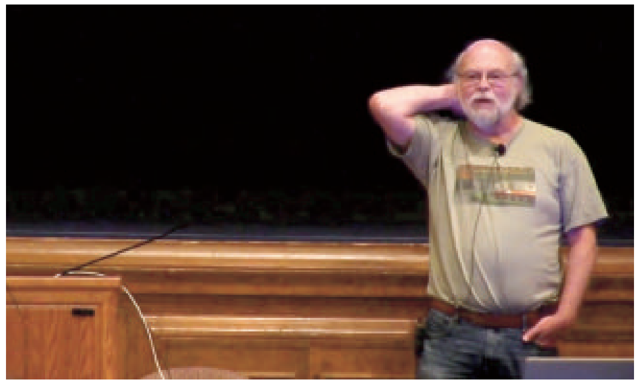
A Variety of Applications

Scientific/environmental. Liquid Robotics' Wave Gliders operating at the air-sea interface can collect atmospheric and oceanographic data while also providing real-time telemetry to undersea sensors.

Commercial/industrial. Wave Gliders can support all phases of offshore energy projects, from exploration to production to long-term monitoring.

Government/defense. Wave Gliders can be used in a wide variety of applications, from port and harbor security to persistent surveillance and real-time subsea-to-air communications.

The desktop interface will use World Wind maps to animate each phase of a mission, from planning the course to monitoring day-to-day performance to reviewing what happened at a particular juncture. However, for the most part, the Wave Gliders are self-sufficient. "They go from one waypoint to another, and so long as nothing exceptional happens they just do their thing," explains Gosling. "However,



Gosling presented "Robots and Water and Whales, Oh My!" at the Bay Area All Java Event in July.

every now and then if the craft is going through an area of intense ocean currents, or there is a ship on a collision course, they'll send a message to the pilots on their iPhones and ask them to intervene to set a different course."

WHY JAVA?

Gosling likes using Java due to its exceptional performance and reliability. He says it's great for the cloud-based components and also ideal for analyzing telemetry data, thanks to its ability to handle complex data structures. "These are not Web applications, so it's not about generating HTML code," he adds. "It's more about the automatic transmission and analysis of data from remote sources."

Due to the exorbitant cost of satellite communications from the open ocean, Gosling contends with a problem that has almost vanished from every other programming domain: extremely limited bandwidth. Everything sent across the satellite link has to be tightly encoded and compressed.

"The physics of our universe is driven by the cost of networking," he says. "Data delivery between the craft and shore is typically through satellite, which costs about $1 per kilobyte. We can occasionally use cheaper and faster networks, but overall, Iridium is our main form of communication on the ocean. So, for us to generate a terabyte of data would cost a billion dollars in network fees."

This problem is reduced when a craft is close to shore or in the vicinity of a drilling rig that has a wide-area communications hub. It's also not a factor if the data is not needed immediately, because it can be cached onboard and downloaded at a future date. However, for most data transmission and analytic functions, economy is essential.

Gosling uses JDK 7 for the majority of this development work, and he is a heavy user of NetBeans, a framework for Java desktop applications. He is looking forward to using JDK 8, the prototype reference implementation of Java SE 8. Project Lambda is particularly interesting.

"The robot is really a platform for all types of test and measurement instruments, whether analyzing water chemistry or listening for whales," Gosling sums up. "Whatever the instrument, it's connected to the onboard computers." </article>

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

ART BY I-HUA CHEN

CELEBRATING 10 YEARS

DUKE'S CHOICE AWARDS

This year's awards include the first-ever Community Choice Award, two user groups, and the United Nations' refugee agency.

BY PHILIP GILL

At first glance, the list of this year's Duke's Choice Award winners reads like a diverse group of unrelated open source projects, businesses, user groups, and individuals whose only common trait is Java. While that is certainly true, a closer reading of this year's winners of the Duke's Choice Awards—the Java community's equivalent of the Oscars—tells us not just about some of the latest technological innovations in Java but about how the Java community has grown and changed over the years as well.

Historically, Duke's Choice Award winners were chosen on the basis of technical innovation, but this year the judges decided to broaden their criteria. In honoring two Java user groups (JUGs)—a first in the Duke's Choice Awards' 10-year history—the awards

DUKE'S CHOICE AWARD WINNERS 2012

(in alphabetical order by organization name)

AgroSense

Hadoop
Apache Software Foundation

JDuchess

Jelastic, Inc.

Liquid Robotics

London Java Community

MICE
North Atlantic Treaty Organization

Parleys.com

Ram Kashyap
Student Nokia Developer Group

Level One Registration Tool
United Nations High Commissioner for Refugees

JDUCHESS
"We provide a platform through which women can connect with each other and get involved in the great Java community."
—Régina ten Bruggencate

drive home the fact that Java remains a user-driven, rather than vendor-driven, phenomenon.

"This year we had more freedom to make decisions," explains John Yeary, one of this year's judges and the leader of the Greenville (South Carolina) Java Users Group (GreenJUG). "We decided as a whole that a couple of winners were worthy of awards not just based on their technology innovation, but by being innovative in their approach to making the community better."

Although the two winning user groups don't develop code, Yeary explains, they do foster the growth and development of those who do. The **London Java Community** (LJC) and its users have been active in the OpenJDK, the Java Community Process (JCP), and other efforts. (LJC representative Ben Evans is the subject of an in-depth interview in this issue—see "JCP Executive Series: A Conversation with Ben Evans," on page 12.)

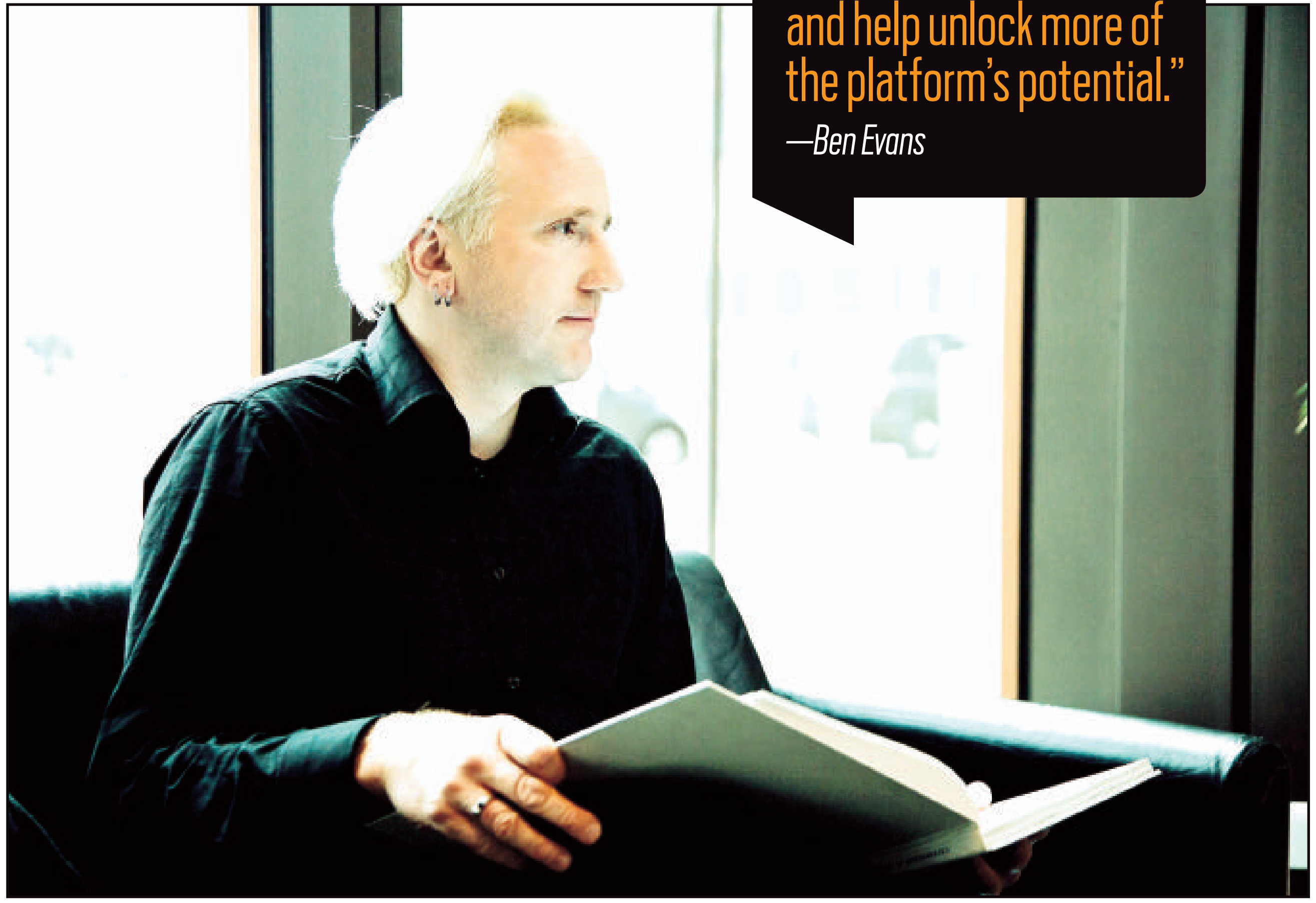
Rather than focus on a specific geographic area like most JUGs, **JDuchess** fosters the participation of women in the Java community worldwide. JDuchess has more than 500 members in 60 countries, according to Régina ten Bruggencate, who runs the group with founder Linda van der Pal.

"We provide a platform through which women can connect with each other and get involved in the great Java community," says ten Bruggencate. "We welcome all women who are interested in Java technology, including developers, architects, testers, business analysts, managers, and others. We believe that women have an essential role to play in the future of IT as it evolves the global economy."

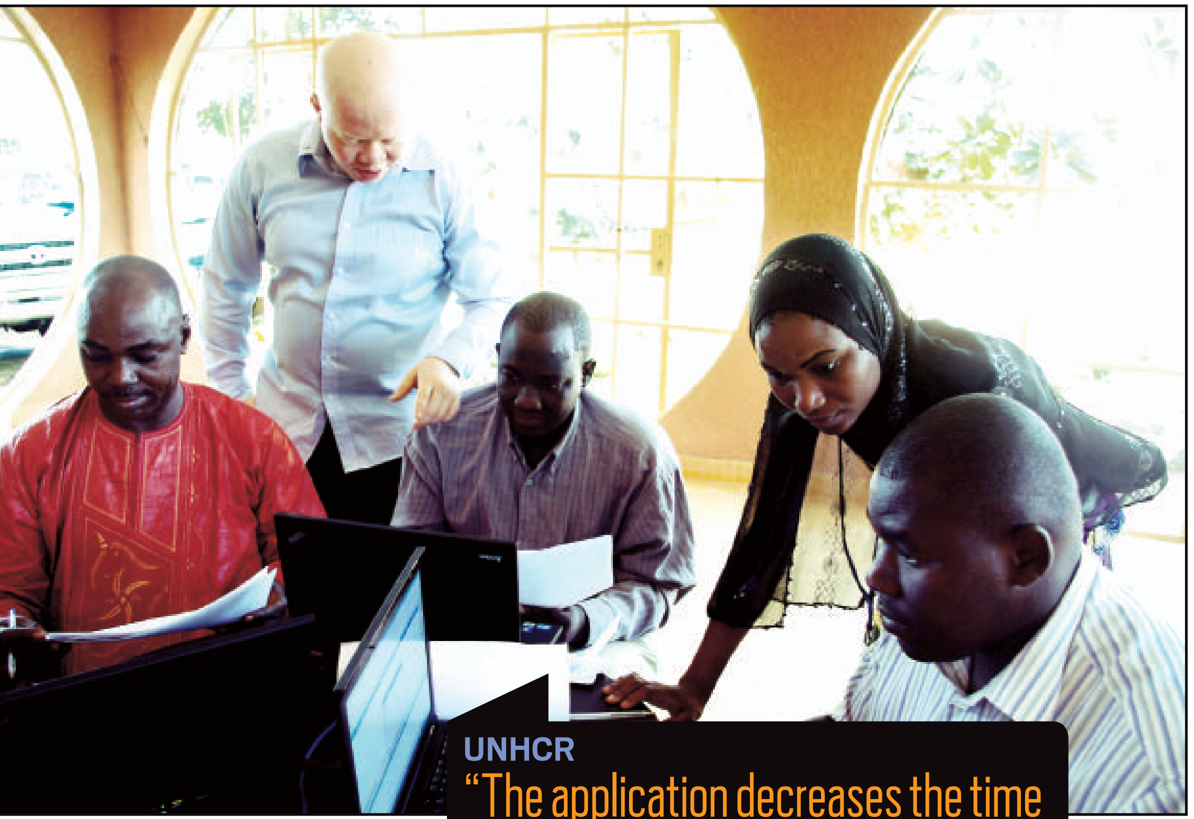
A HEART AND A SOUL

While there's no question that Java has "a head for business," this year's winners are using the technology to tackle some of the pressing human-

LONDON JAVA COMMUNITY
"I personally wanted our JUG to become more of a community participant. I wanted to participate and help unlock more of the platform's potential."
—Ben Evans



PHOTOGRAPHS BY TON HENDRIKS, JOHN BLYTHE



UNHCR
"The application decreases the time needed to give the first assistance to refugees and planning to ameliorate the response."
—*Stanyslas Matayo* (pictured standing, with team)

PHOTOGRAPH COURTESY OF UNHCR

itarian and environmental challenges of the day. The **United Nations High Commissioner for Refugees** (UNHCR) is on the front lines of crises around the world, from civil wars to natural disasters. To help facilitate its mission of humanitarian relief, the UNHCR has developed a light-client Java application on the NetBeans platform.

First deployed this year in the Western African republics of Mali and Niger, the Level One registration tool enables the UNHCR to collect information on the number of refugees and to plan the distribution of water, food, housing, healthcare, and other aid in the field, and combines those capabilities with geocoding information from various sources. This enables the UNHCR to deliver the appropriate kind and amount of assistance where it is needed. "The application decreases the time needed to give the first assistance to refugees and planning to ameliorate the response," explains Stanyslas Matayo, team leader on the Level One development project.

"Level One registration helps to solve the issues of registering and locating families and friends," says GreenJUG's Yeary. "The impact on humanity from this software is clear. There is no service higher than helping your fellow man. It helps me sleep better at night knowing that people are out there working on projects like these."

Community Choice: NATO's MICE

The first-ever Community Choice Award goes to the **MASE Integrated Console Environment (MICE)** in use at the North Atlantic Treaty Organization (NATO). Built in Java on the NetBeans platform, MICE provides a high-performance visualization environment for conducting air defense and battle-space operations.

According to Angelo D'Agnano, a software architect at the NATO Programming Centre, MICE updates NATO's Multi-AEGIS Site Emulator (MASE) console, which was built in the 1990s using C/C++ and Motif. Java was chosen because it is already used in many components of NATO's Air Command and Control System (ACCS).

MICE relies on two external libraries: LuciadMap, a set of software components for high-performance visualization for the system's geographic information display, and the NetBeans platform for the application framework. "While the usage of the LuciadMap has helped to achieve the required real-time performance for the display," D'Agnano writes in a post in the NetBeans Zone of the DZone developer site, "the NetBeans RCP [rich client platform] allowed the development team to save significant time; the ready to use solutions, design patterns and guidelines helped to achieve a solid and consistent design of the application."

In May 2012, the NATO Programming Centre delivered the first operational version of the MICE console to 60 locations in 20 countries.

Improving farming methods to feed a hungry world is the goal of **AgroSense**, an open source farm information management system built in Java and the NetBeans platform. Lead developer Timon Veenstra says AgroSense enables farmers, agribusinesses, suppliers, and others to develop application modules that will

AGROSENSE
"AgroSense provides a rich client that enables farmers to work offline when in the field or when the connection is bad, and full connectivity when a broadband connection is available."
—*Timon Veenstra*

PHOTOGRAPH BY TON HENDRIKS

easily exchange information through a common underlying NetBeans framework. These modules will be sold through an online app store, similar to Apple's iTunes. Nonprofit modules can be provided to the farmers free of charge. Precision agriculture support modules will allow farmers to get maximum crops with the least amount of fertilizer and chemicals.

Veenstra says that current rival solutions either "do not communicate with the rest of the world" or require high-bandwidth internet connections not always available in rural areas. "AgroSense provides a rich client that enables farmers to work offline when in the field or when the connection is bad, and full connectivity when a broadband connection is available."

Another Duke's Choice Award winner, **Liquid Robotics**, is an ocean data services provider whose Wave Glider technology collects information from the world's oceans for application in government, science, and commercial applications. Liquid Robotics boasts Java pioneer James Gosling as its chief software architect, and is the subject of this issue's cover story, "Java at Sea," on page 17.

TECHNOLOGY LEADERS

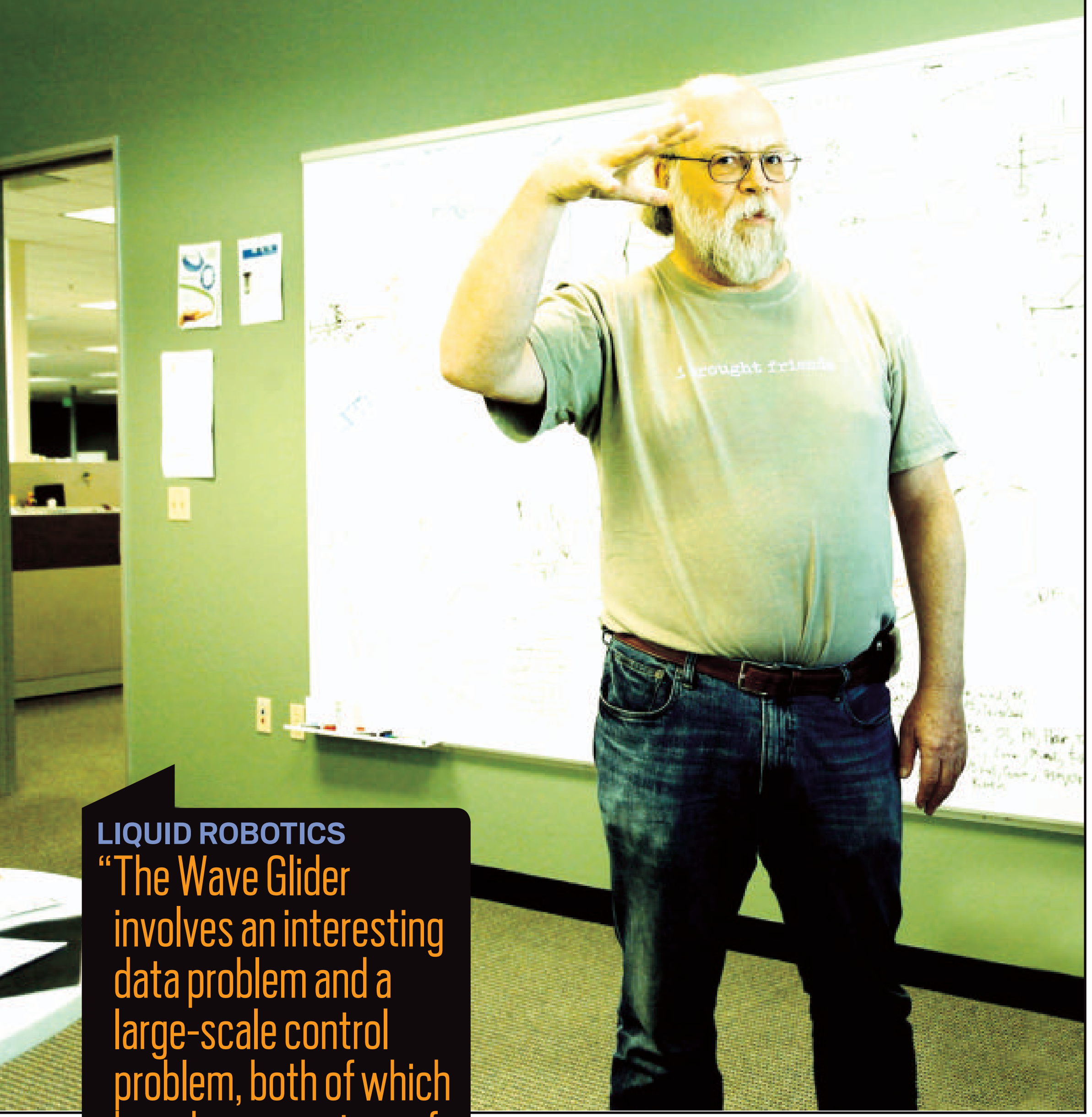
Not surprisingly, this year's awards also show us that Java is at the forefront of today's most-important technological challenges. Moving existing Java applications to the cloud can be a daunting task, but startup **Jelastic, Inc.**, offers the first Java platform as a service (PaaS) that enables existing Java applications to get up and running in the cloud without code changes or lock-in, according to Judah Johns, the Palo Alto, California, firm's chief evangelist. In addition, the Jelastic platform offers unlimited scalability, also without changes to the existing code.

"The Jelastic platform is the only standards-based Java hosting platform and the first to be offered through service providers around the globe," says Johns.

Handling *big data*—extremely large, complex data beyond the ability of traditional data management tools—is a pressing concern for all types of organizations. The Apache Software Foundation's **Hadoop** project, written in Java, provides a framework for distributed processing of large data sets across clusters of computers, from a few servers to thousands of machines.

"Hadoop permits institutions to benefit from much more of the data they generate," explains Doug Cutting, chairman of the Apache Software Foundation and founder of the Hadoop project. "With Hadoop they can afford to store and analyze data that they were previously discarding. Harnessing more data lets folks better understand and improve their business."

"Java is the primary language of the Hadoop ecosystem," Cutting continues. "And Hadoop is the de facto stan-



LIQUID ROBOTICS
"The Wave Glider involves an interesting data problem and a large-scale control problem, both of which have been passions of mine for years."
—*James Gosling*

PHOTOGRAPH BY BOB ADLER

dard operating system for big data. So, as the big data trend spreads, Java spreads too."

Two other winners are adding novel twists to existing technologies. E-learning specialist **Parleys.com**, based in Brugge, Belgium, uses Java technologies to bring online classes and full IT conferences to desktops, laptops, iPads, and Android and BlackBerry PlayBook devices. Founder and CEO Stephan Janssen says Parleys has hosted more than 1,700 conferences—including Devoxx and JavaOne—for more than 800,000 unique visitors. "Parleys allows users to view a presentation on their desktop or download it on their device for offline enjoyment—whenever and wherever they want," says Janssen.

Finally, this year's student winner, **Ram Kashyap**, has already graced the pages of *Java Magazine*. As the founder and president of the Student Nokia Developer group, he was profiled in the cover story in the March/April 2012 issue of *Java Magazine*, "The New Java Developers." Since then, Kashyap has maintained a hectic pace, graduating from the People's Education Society Institute of Technology in Bangalore, India, while working on a Java mobile startup and training students on Java ME. </article>

Philip Gill is a San Diego, California–based writer and editor who has been following Java for 20 years.

JUDGES AND PROCESS

The winners of the 10th annual Duke's Choice Awards were selected in a three-part process. All members of the Java community were first invited to submit nominations to this year's judges. In the first two rounds, the judges selected 9 winners from 68 submissions and nominated 6 candidates for the first-ever Community Choice Award. In the final round, Community Choice Award nominees were posted on Java.net, and all members of the Java community were invited to vote for their favorite. The winner, the MICE project from the NATO Programming Centre, received 41 percent of the vote.

This year's judges were

Yara Senger, SouJava

John Yeary, president and founder, Greenville Java Users Group

Glen Peterson, Greenville Java Users Group member and CTO, PlanBase Inc.

Martijn Verburg, London Java Community (who abstained from the vote electing his JUG)

Michelle Kovac, Java marketing and operations

Arun Gupta, Java evangelist and GlassFish community member

Sharat Chander, Java evangelist team manager

The 10 winners will be honored at JavaOne September 30 through October 4 in San Francisco, California.

MICHAEL KÖLLING

BIO

Part 2

Learning About Object Interaction with BlueJ

Abstraction and modularization in object-oriented programming

In the last issue of *Java Magazine*, I gave you a quick overview of the BlueJ environment and presented the tools that it offers for teaching programming concepts to beginners. This time, we will discuss a small programming example to see how BlueJ can help us understand some fundamental, but often difficult to grasp, programming concepts.

We will again use a practical programming example, which you can download and then play along with as we go.

The Clock Example

The project we will use to discuss the interaction of objects is a display for a digital clock. The display shows hours and minutes, separated by a colon (see **Figure 1**). For this exercise, we will first build a clock with a European-style 24-hour display. Thus, the display shows the time from 00:00 (midnight) to 23:59 (one minute before midnight). It turns out, upon closer inspection, that building a 12-hour clock is slightly more difficult, so we will leave that until the end of this article.

Abstraction and Modularization

A first idea for many beginning programmers might be to implement the whole clock display in a single class. That is, after all, what classes are for: they represent things, and a clock is a thing.

However, here we shall approach this problem slightly differently. We will see whether we can identify subcomponents in the problem that we could turn into separate classes. The reason is *complexity*. As we progress in our programming practice, the programs we build will get more and more complex. Trivial tasks that we encounter as initial exercises can be solved as a single problem. You can look at the complete task and devise a solution using a single class. For more-complex problems, that is too simplistic. As a problem grows larger, it becomes increasingly difficult to keep track of all details at the same time.

The solution we use to deal with the complexity problem is *abstraction*. We divide the problem into subproblems and divide those again into more subproblems, and so on, until the individual problems are small enough to be easy to deal with. Once we solve one of the subproblems, we do not think about the details of that part anymore, but we treat the solution as a single building block for our next problem. This technique is sometimes referred to as *divide and conquer*.

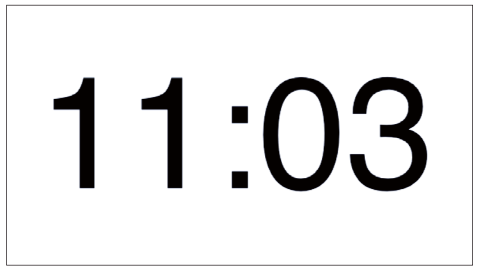


Figure 1

Let's discuss this with an example. Imagine engineers in a car company designing a new car. One engineer might think about the parts of the car, such as the shape of the outer body, the size and location of the engine, the number and size of the seats in the passenger area, the exact spacing of the wheels, and so on. Another engineer, on the other hand, whose job is to design the engine (well, that's a whole team of engineers in reality, but we can

PHOTOGRAPH BY JOHN BLYTHE

simplify a bit here for the sake of the example), thinks of the many parts of an engine: the cylinders, the injection mechanism, the carburetor, the electronics, and so on. She will think of the engine not as a single entity, but as a complex work of many parts. One of these parts might be a spark plug.

Then there is an engineer (maybe he works in a different company) who designs the spark plugs. He will think of the spark plug as a complex artifact of many parts. He might have done complex studies to determine exactly what kind of metal to use for the contacts or what kind of material and production process to use for the insulation.

The same is true for many other parts. A designer at the highest level will regard a wheel as a single part. Another engineer much further down the chain might spend her days thinking about the chemical composition to produce the right materials for making the tires. For the tire engineer, the tire is a complex thing. The car company will just buy the tire from the tire company and then view it as a single entity. This is abstraction.

The engineer in the car company *abstracts from* the details of the tire manufacture to be able to concentrate on the details of the construction of, say, the wheel. The designer designing the body shape of the car abstracts from the technical details of the wheels and the engine to concentrate on the design of the body (he will just be interested in the size of the engine and the wheels).

The same is true for every other component. While someone might be concerned with designing the interior passenger space, someone else might work on developing the fabric that will eventually be used to cover the seats.

The point is, if viewed in enough detail, a car consists of so many parts that it is impossible for a single person to know every detail about every part at the same time. If that were necessary, no car could ever be built.

The reason why cars are built successfully is that the engineers use *modularization* and abstraction. They divide the car into independent modules (wheel, engine, gearbox, seat, steering wheel, and so on) and get separate people to work on separate modules independently. When a module is built, they use abstraction. They view that module as a single component that is used to build more-complex components.

Modularization and abstraction thus complement each other. Modularization is the process of dividing large things (problems) into smaller parts, while abstraction is the ability to ignore details to focus on the bigger picture.

Modularization and Abstraction in Software

Modularization and abstraction are also used in software development. To maintain an overview in complex programs, we try to identify subcomponents that we can program as independent entities. Then we try to use those subcomponents as if they were simple parts without being concerned about their inner complexities.

In object-oriented programming, these components and subcomponents are *objects*. If we were trying to construct a car in software using an object-oriented language, we would try to do what the car engineers do. Instead of implementing the car in a single, monolithic object, we would first construct separate objects for an engine, gearbox, wheel, seat, and so on, and then assemble the car object from those smaller objects.

Identifying what kinds of objects (and with these, classes) you should have in a software system for any given problem is not always easy. In this article, I will show you how BlueJ can help you build, test, and experiment with these kinds of components.

Now, back to our digital clock.

Examining the Project

Open the clock-display project, which you downloaded, in BlueJ. Initially, you will see a very simple class diagram (see **Figure 2**). We can see that our project consists of two classes named NumberDisplay and ClockDisplay.

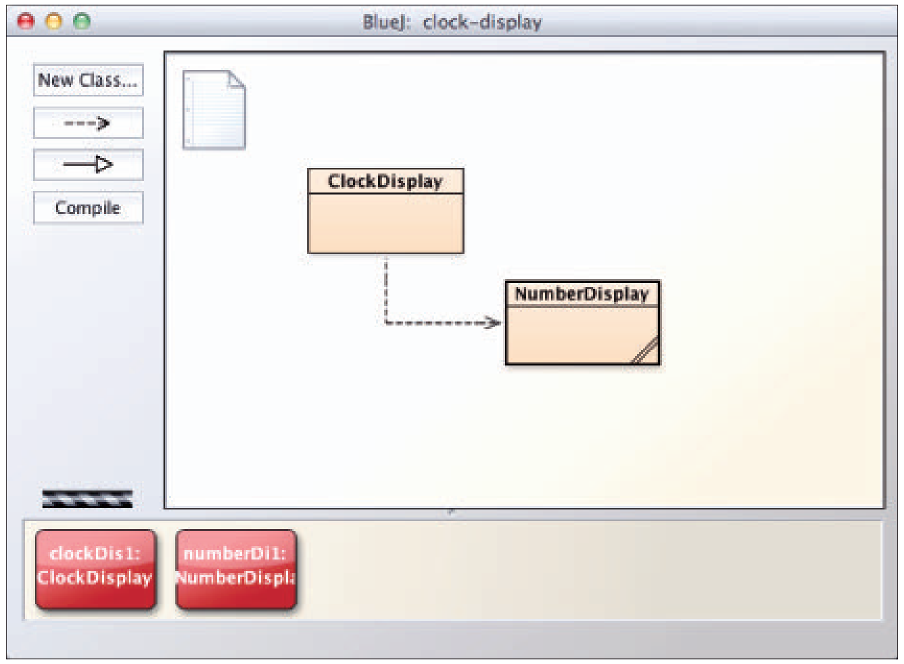


Figure 2

Figure 3

NumberDisplay is a class that represents a single, two-digit display of a number (see **Figure 3**). We will see that we can use it to build our clock display. In fact, we can use *two instances* of the number display: one for the hours and one for the minutes. This way, we are breaking down our problem of building a clock display into smaller, easier-to-solve problems.

We first build a class for a two-digit number display, and then we build the clock display by taking two number displays (one for the hours and one for the minutes) and sticking them together. Each of the subproblems is easier to solve than the whole.

The NumberDisplay class represents objects that can store a number value up to a given limit. It can be incremented, and when the value reaches that limit, it rolls around back to zero. We will use one of these objects with a limit of 60 for the minutes and another one with a limit of 24 for the hours.

In BlueJ, we can inspect the object in two different ways: we can read the source code, or we can experiment with the object by invoking its methods. Let's start with the latter.

Create an instance of the class NumberDisplay by right-clicking the NumberDisplay class in your diagram and selecting the first entry in the menu, which is titled **new NumberDisplay(int rollOverLimit)** (see **Figure 4**). By using this menu command, you are invoking the class's constructor to create an object.

A dialog box will pop up and prompt you to enter a name for the instance and a value for the constructor parameter. Here we can specify the limit at which we want the display to roll over. Let's leave the instance name as it is suggested, and use 60 for the parameter. Once you click **OK**, a NumberDisplay object will appear on the object bench (as shown in **Figure 2**).

We can now experiment with the NumberDisplay by right-clicking the object and trying out its methods (see **Figure 5**).

We can see that the object has four methods. Try them out by selecting each one of them. You will see that they do the following:

- setValue(int replacementValue) lets you specify a value for this display.
- getValue() returns the current value.
- getDisplayValue() returns the same value (as a String), padded with a leading zero to ensure that the result is always a two-digit number (for example, "04" instead of "4").
- increment() increments the value by one.

Another way to experiment with the NumberDisplay object is to open the object inspector by using the **Inspect** function from the object's menu (see **Figure 6**).

Here, we can see that the NumberDisplay object holds two fields: one for the current value and one for its limit. Leave the object inspector open, and call some of the object's methods again. Now you can see the value change and observe the effects of the methods. An interesting thing to try is to call the increment method when the value is just below the limit. Try it out!

Another, more traditional way to inspect a class is to read its source code. Open the source code for the NumberDisplay class (by double-clicking the class icon) and read through it. You will see how each of the four methods was implemented.

Examining ClockDisplay

Through experimentation with the NumberDisplay object, we can get a feeling for the behavior and functionality of that object. Now we will look at how the ClockDisplay class uses two NumberDisplay objects to create a clock display.

Let's start again by creating an

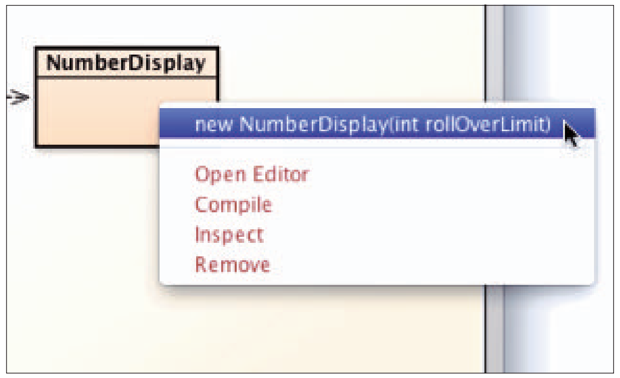


Figure 4

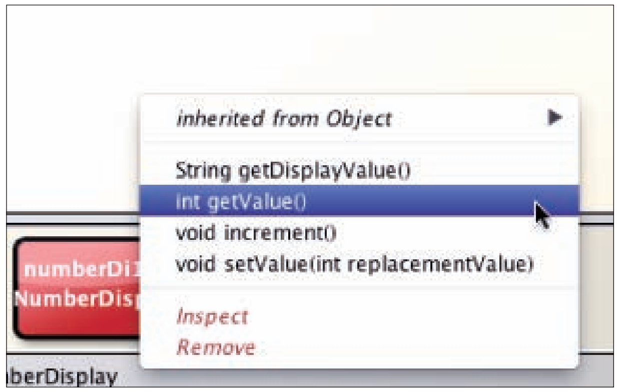
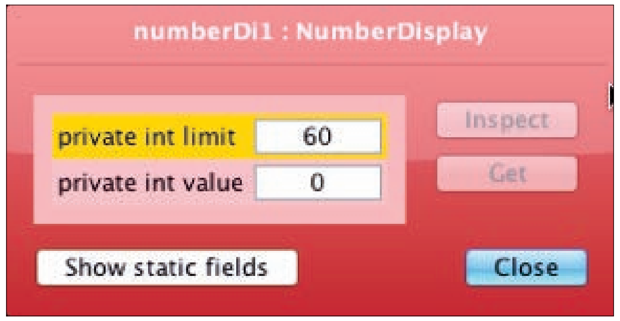


Figure 5



numberDi1 : NumberDisplay

private int limit 60
private int value 0

Figure 6

object and experimenting with its methods:

- Create an object of class ClockDisplay by right-clicking the class and selecting its constructor. You will see that there are two constructors—one with parameters and one without. Try both.
- Right-click the ClockDisplay object on the object bench and try out its methods. You will see that there are three: getTime, setTime, and timeTick.
- Open the object inspector for the ClockDisplay object (see **Figure 7**) by choosing **Inspect** from the menu, and observe its fields while you call its methods.

If you do this, you will observe a number of things:

- The ClockDisplay object has two fields of type NumberDisplay, named hours and minutes, for storing its time.
- The displayString field in the ClockDisplay object shows the current time of the clock. We use this in our project to simulate the display of our clock. In a real clock, this is the time that would be shown on its display.
- The timeTick method does not seem to do anything.

The last point, the lack of action when we call the timeTick method, has its roots in a simple fact: the project implementation is not finished. In a real clock, the timeTick method would be called regularly (once a minute) by the clock timer to advance the clock. We now want to implement this behavior.

Inspecting the Source Code

Let's start by inspecting the ClockDisplay source code. Open its editor by double-clicking the ClockDisplay class icon.

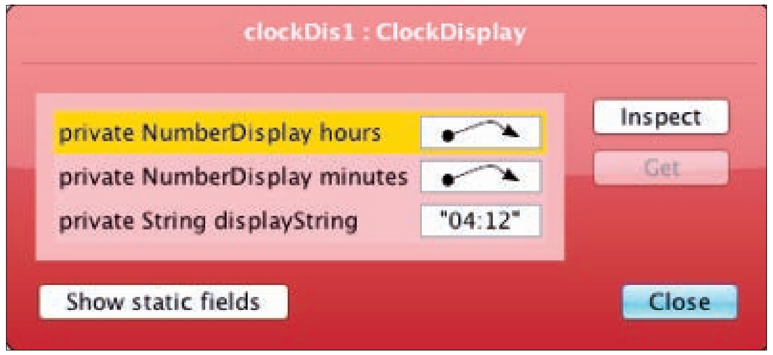
Looking through the code, we can observe the following interesting facts. Three fields are declared, two of type NumberDisplay and one of type String:

```
private NumberDisplay hours;
private NumberDisplay minutes;
private String displayString;
```

The first two hold the NumberDisplay objects for the hours and minutes, and the last represents the current display of the clock. The class's constructor initializes the three fields:

```
public ClockDisplay()
{
 hours = new NumberDisplay(24);
 minutes = new NumberDisplay(60);
 updateDisplay();
}
```

Examine the source code of the other methods to familiarize yourself with how they work. You will see that their implementation is not very complex. You will also see that the method timeTick is empty, explaining why it does not yet work. We will now implement this method.



private NumberDisplay hours
private NumberDisplay minutes
private String displayString "04:12"

Figure 7

The purpose of the timeTick method is to advance the clock by one minute. We can achieve this by adding the following line of code to its body:

```
minutes.increment();
```

This is almost all that's required, except for one thing: we have to deal with the case where the hours are also incremented (for example, the time advances from 03:59 to 04:00). We know that this should happen when the minutes roll over to zero, so we can add the following:

```
if(minutes.getValue() == 0) {
  hours.increment();
}
```

And after doing this increment, we also have to update the display value:

```
updateDisplay();
```

This is the whole implementation. After adding this to your class, your timeTick method should look like the one shown in **Listing 1**.

Do It Yourself

Now try it out yourself. Type the code as we have shown it here, and compile the class (click the **Compile** button either in the editor window or in the main window). If you get any errors, carefully inspect your code and fix them.

Now create a ClockDisplay object again, open its object inspector, and experiment with its methods. In particular, try the newly implemented timeTick method. You will see that it should work. Try it also at a point where the hour changes and make sure that works as well.

And Then Do More . . .

I have shown you how to examine an existing project with BlueJ and how to make a small improvement. As a programming exercise, you can now try several further improvements, for example:

- Display a warning when a user

tries to set the time to an invalid value.
- Add seconds to the clock display.
- Modify the clock to show a U.S.-style 12-hour display (04:23 p.m.) instead of the European-style 24-hour version (16:23).

Each of these tasks makes a nice little programming exercise for beginners who know the basics and want to practice their programming skills.

LISTING 1

```
/**
 * This method should get called once every minute - it makes
 * the clock display go one minute forward.
 */
public void timeTick()
{
   minutes.increment();
   if(minutes.getValue() == 0) {  // it just rolled over!
      hours.increment();
   }
   updateDisplay();
}
```

Download all listings in this issue as text

Conclusion

Today, I have presented a small programming example that started with examining a given body of code, progressed to gaining an understanding of existing functionality, and finished by extending the project with new functionality. We can clearly see now how BlueJ helps in the learning of fundamental concepts.

For teachers and learners, several aspects of the BlueJ environment are crucial to the process of learning:
- We can easily see the classes involved in the project and their relationships.
- We can examine any class by interactively creating an instance of the class and calling its methods. This gives us a good understanding of the behavior of individual objects.
- We can look into objects using the object inspector.
- We can easily test and experiment using the object bench.
- We can concentrate on the object structure and logic of the program first without the distraction of a GUI or a main method. This teaches good object-oriented principles before tackling tedious detail. (Of course, GUIs and main methods can be added as well.)

If you are a teacher, or someone who wants to teach someone object-oriented programming, you probably get the idea. BlueJ offers a unique toolset that is much better suited for learning the fundamentals of programming than big, professional environments, such as NetBeans or Eclipse. It has just the right kind and right amount of tool support needed for programmers in their first year, and developers can then graduate from BlueJ to the professional environments once they have developed a good mental model of object orientation.

So far, I have just shown the basics of the BlueJ tools. In the next episode, we will look at specialized support for testing. </article>

LEARN MORE

- Java SE 7 API
- BlueJ Website

MAKE THE FUTURE JAVA

Java™

FIND YOUR JUG HERE

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

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

LEARN MORE

ORACLE

PHOTOGRAPH BY
ALLEN MCINNIS/GETTY IMAGES

MAX BONBHEL

BIO

Part 2

Introduction to Web Service Security from Server to Client

Handle transport security using SSL to increase the security of Web services.

In Part 1 of this three-part series, I showed you how to secure a Web service efficiently both on the server and on the client using Metro, GlassFish, and the NetBeans IDE.

Now in Part 2, we will see in detail how to use NetBeans tools and GlassFish default digital certificates to configure the transport security mechanism to protect the AuctionApp application during transport by using Secure Sockets Layer (SSL) for authentication. The goal of this article is to show how easy it is to add transport-layer security to protect the data transiting between the client and the server using SSL via the secure HTTP transport, HTTPS.

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

What Is the Transport Security Mechanism?

Transport security is a mechanism for securing data that is circulating between a point A and a point B. Often used with SSL, this mechanism can cover several aspects of data confidentiality, and it allows the server and the client to convert a common data encryption algorithm using a certificate exchange.

Prerequisites

Download the following software, which was used to develop the application described in this article:

- NetBeans IDE 7.2 (available for download here)
- GlassFish 3.1.2.2 (available for download here)
- Metro 1.3 or higher (included in NetBeans)

Note: This article was tested using the latest version of the NetBeans IDE (version 7.2, at the time this article was written).

Overview of Adding Transport Security to the Web Service

What we are going to do is secure the bytes of data that are transmitted or received between the client and the server over an SSL-protected session.

We will perform the following tasks:

- Secure the Web service by adding the transport security mechanism (SSL) and configuring the service to use HTTPS
- Create and configure a new client that references the Web service by specifying the secure Web Services Description Language (WSDL) file
- Test the secure Web service

Note: The application we are going to secure is an online auction place (like eBay) that we created in a previous series of articles ("Introduction to RESTful Web Services"). Sellers post their items in listings, and buyers bid on the items. A seller can post one or many items, and a buyer can bid on one or many items.

Specifically, we are going to limit access to the Java API for XML Web Services (JAX-WS) Web service that extrapolates the amount of a bid.

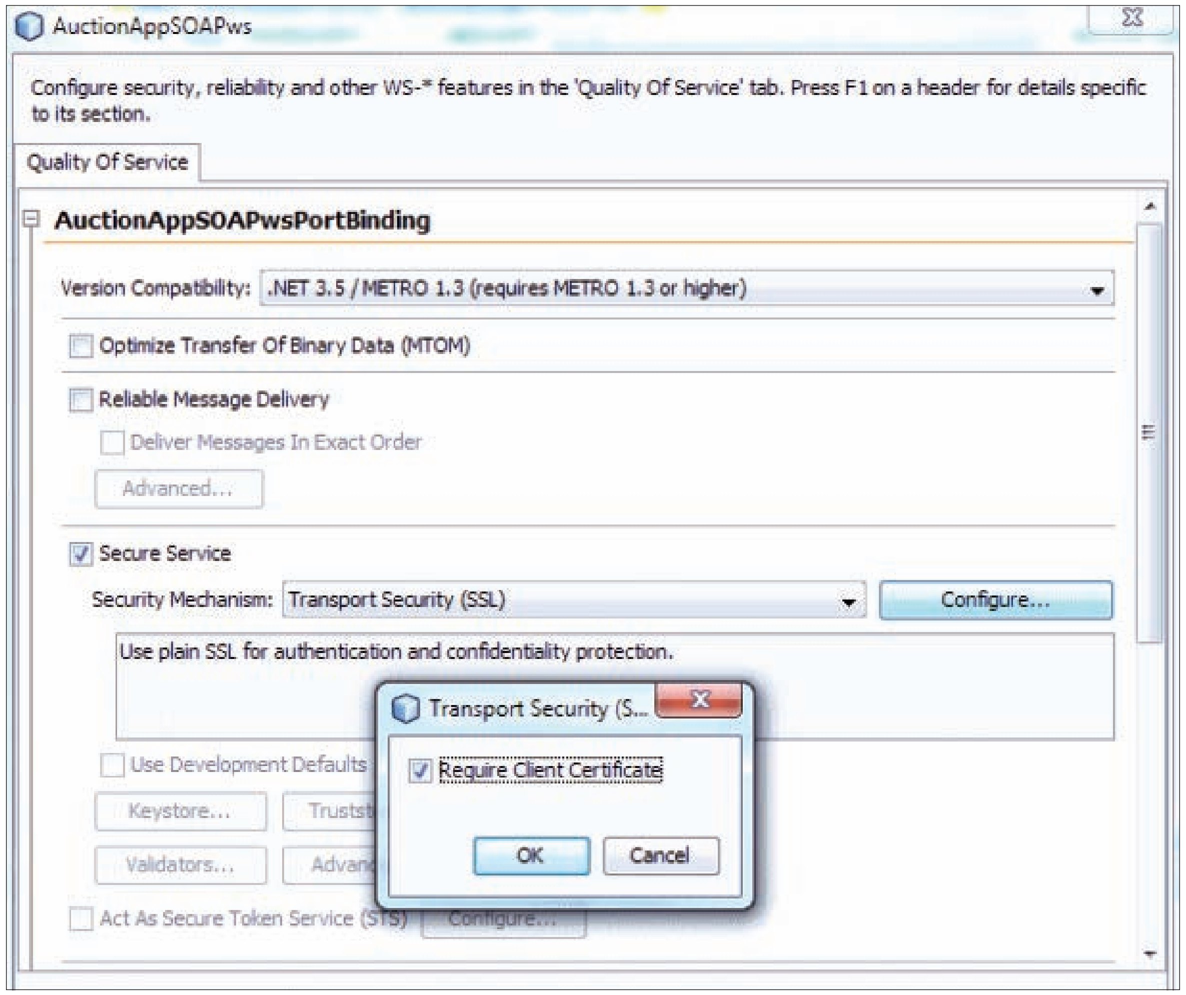
Add Transport Security to the Web Service

Let's add transport security to secure the Web service, in only a few minutes, using the NetBeans IDE.

1. Add a security mechanism called Transport with Symmetric Key to the AuctionApp application:
   - **a.** Open the AuctionApp project in NetBeans IDE 7.2 or higher.
   - **b.** Expand the **Web Service** node of the AuctionApp project and right-click the **AuctionAppSOAPws** node; then select **Edit Web Service Attributes**.
   - **c.** Under the **Quality Of Service** tab, expand the **AuctionApp-SOAPwsPortBinding** section. (See **Figure 1**.)
   - **d.** Make sure the **Reliable Messaging Delivery** option is *deselected*.
   - **e.** Select **Secure Service** and then select **Transport Security (SSL)** from the Security Mechanism list.
   - **f.** Click **Configure** and select the **Require Client Certificate** option.
   - **g.** Select **Use Development Defaults**.
   - **h.** Click **OK**.

   At this point, NetBeans creates a new Web Services Interoperability Technologies (WSIT) configuration that contains the security elements within the wsp:Policy tags of the wsit-com .bonbhel.oracle.auctionApp .AuctionAppSOAPws.xml file. The file is located in the Web Pages/WEB-INF node of the AuctionApp project.
2. Configure the service to use HTTPS:

   To be sure the application uses SSL, we are going to specify the security requirements, such as the URL pattern to be protected and the Transport Guarantee option in the deployment descriptor file (Web.xml).
   - **a.** Expand the **Web Pages | WEB-INF** node of the AuctionApp project and double-click the Web.xml file.
   - **b.** From the **Security** tab, click **Add Security Constraint** to create specific constraints for AuctionAppSOAPws.
   - **c.** For **Display Name**, type a name such as SSL transport for AuctionAppSOAPws.
   - **d.** From the Web Resource Collection, click **Add**; then type Secure Area in the **Resource Name** field, as shown in **Figure 2**.
   - **e.** Type /AuctionAppSOAPws//* in the **URL Pattern(s)** field.
   - **f.** Select the **Selected HTTP Methods** option and select the **GET** and **POST** checkboxes.
   - **g.** Click **OK**.
   - **h.** Make sure the **Enable Authentication Constraint**


AuctionAppSOAPws

Configure security, reliability and other WS-* features in the 'Quality Of Service' tab. Press F1 on a header for details specific to its section.

Quality Of Service

AuctionAppSOAPwsPortBinding

Version Compatibility: .NET 3.5 / METRO 1.3 (requires METRO 1.3 or higher)

☐ Optimize Transfer Of Binary Data (MTOM)

☐ Reliable Message Delivery

☐ Deliver Messages In Exact Order

Advanced...

☑ Secure Service

Security Mechanism: Transport Security (SSL) — Configure...

Use plain SSL for authentication and confidentiality protection.

☐ Use Development Defaults

Keystore... Truststore... Validators... Advanced...

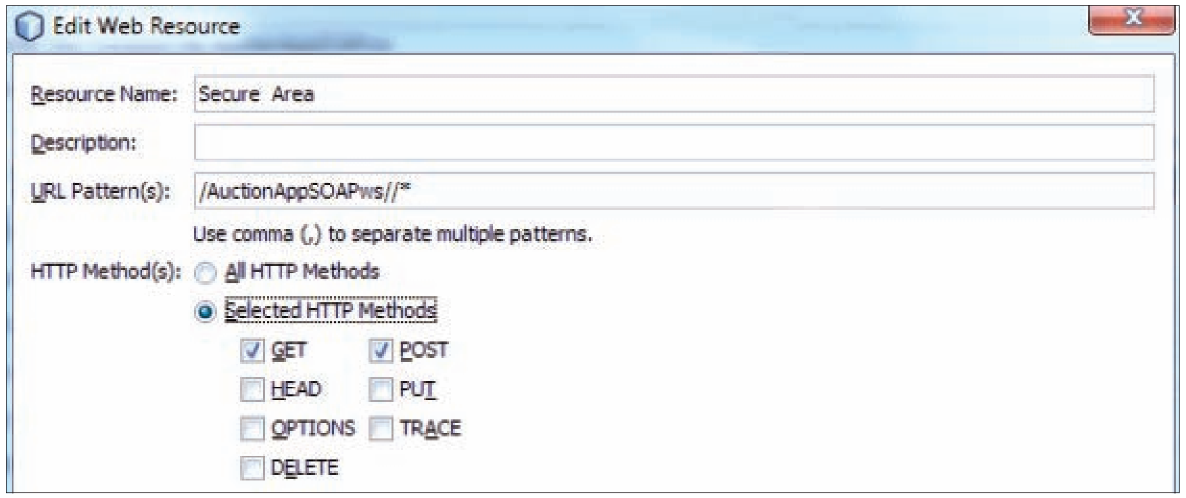
☐ Act As Secure Token Service (STS) — Configure...

Transport Security (S...

☑ Require Client Certificate

OK Cancel

Figure 1


Edit Web Resource

Resource Name: Secure Area

Description:

URL Pattern(s): /AuctionAppSOAPws//*

Use comma (,) to separate multiple patterns.

HTTP Method(s): ○ All HTTP Methods

◉ Selected HTTP Methods

☑ GET ☑ POST

☐ HEAD ☐ PUT

☐ OPTIONS ☐ TRACE

☐ DELETE

Figure 2

option is *deselected*, as shown in **Figure 3**.

i. Select the **Enable User Data Constraint** option if it is not selected and make sure **CONFIDENTIAL** is chosen from the Transport Guarantee list.
NetBeans generates the appropriate configuration in the deployment descriptor.

j. Click the **Source** tab to see all the details.

3. Deploy and test the Web service application:

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

b. Open your browser and type the resource URL https://localhost:8181/AuctionApp/AuctionAppSOAPws?wsdl.
The server presents its certificate, as shown in **Figure 4**.

c. Accept the certificate.
The WSDL file for the application is displayed.

Create and Configure a New Web Client

In this section, we are going to create and secure a new Web service client that references the Web service that we just secured. To do this, we will create the client application.

We will use the Web Service Client wizard provided by NetBeans IDE 7.2 to generate the code and everything we need for looking up a Web service.

Let's code this application in five minutes.

1. Generate the initial NetBeans project:

a. From the File menu, choose **New Project**.

b. From Categories, select **Java Web**.

c. From Projects, select **Web Application**.

d. Click **Next**.

e. Type WebServiceClientSecureSSL for the project name and click **Next**.

f. Make sure the server is specified as GlassFish Server (or similar wording).

g. Click **Finish**.

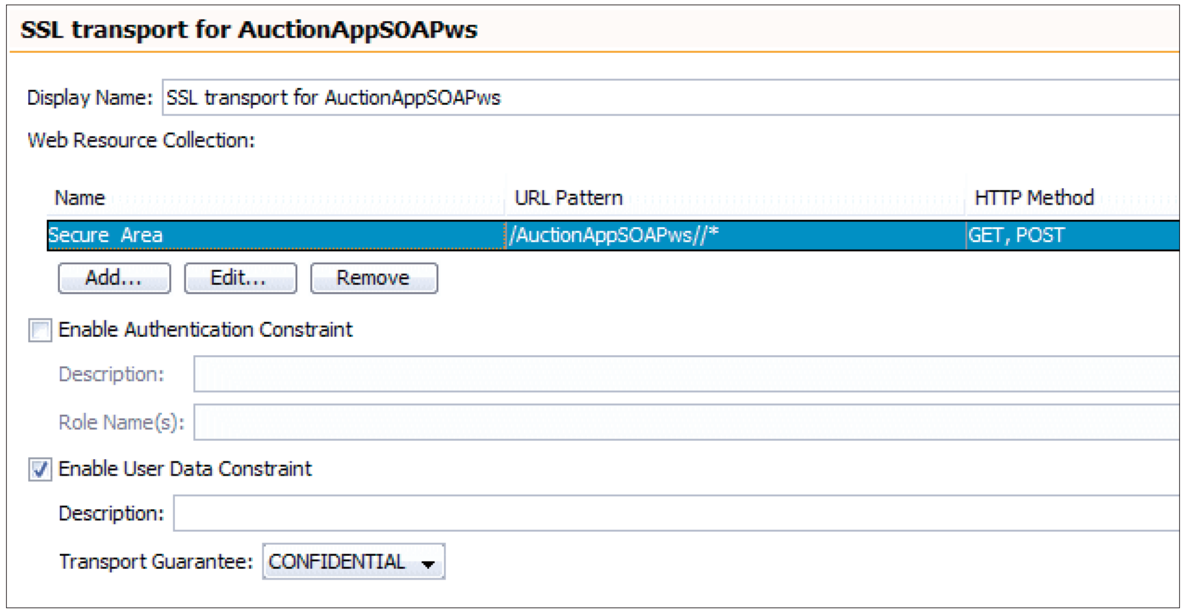
2. Create the Seller entity:

a. Right-click the **WebServiceClientSecureSSL** project and select **New**; then select **Entity class**.

b. Type Seller in the **Class Name** field, type com.bonbhel.oracle.webServiceClientSecureSSL in the **Package** field, and click **Next**.

c. In the Provider and Database screen, select **EclipseLink (JPA 2.0)(default)** from the Persistence Provider list, as shown in **Figure 5**.

d. From the Data Source list, select **jdbc/sample**, which is



SSL transport for AuctionAppSOAPws

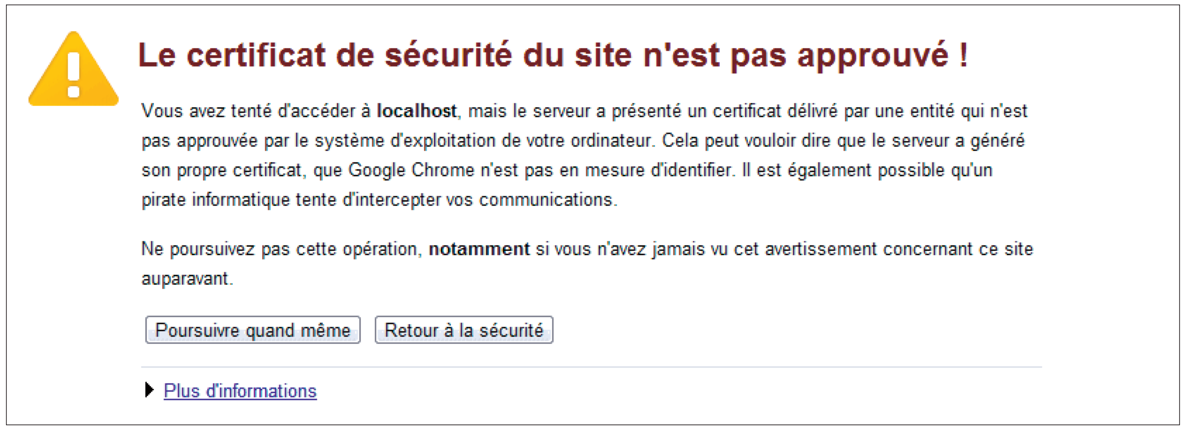
Display Name: SSL transport for AuctionAppSOAPws

Web Resource Collection:

Name	URL Pattern	HTTP Method
Secure Area	/AuctionAppSOAPws//*	GET, POST

☐ Enable Authentication Constraint
Description:
Role Name(s):
☑ Enable User Data Constraint
Description:
Transport Guarantee: CONFIDENTIAL

Figure 3

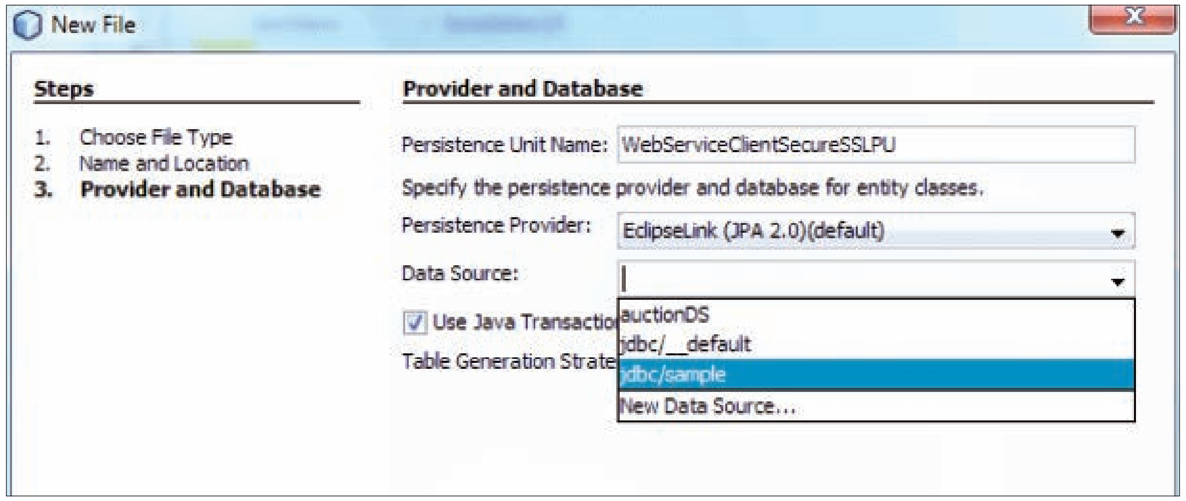


Le certificat de sécurité du site n'est pas approuvé !

Vous avez tenté d'accéder à **localhost**, mais le serveur a présenté un certificat délivré par une entité qui n'est pas approuvée par le système d'exploitation de votre ordinateur. Cela peut vouloir dire que le serveur a généré son propre certificat, que Google Chrome n'est pas en mesure d'identifier. Il est également possible qu'un pirate informatique tente d'intercepter vos communications.

Ne poursuivez pas cette opération, **notamment** si vous n'avez jamais vu cet avertissement concernant ce site auparavant.

Figure 4



New File

Steps
1. Choose File Type
2. Name and Location
3. **Provider and Database**

Provider and Database

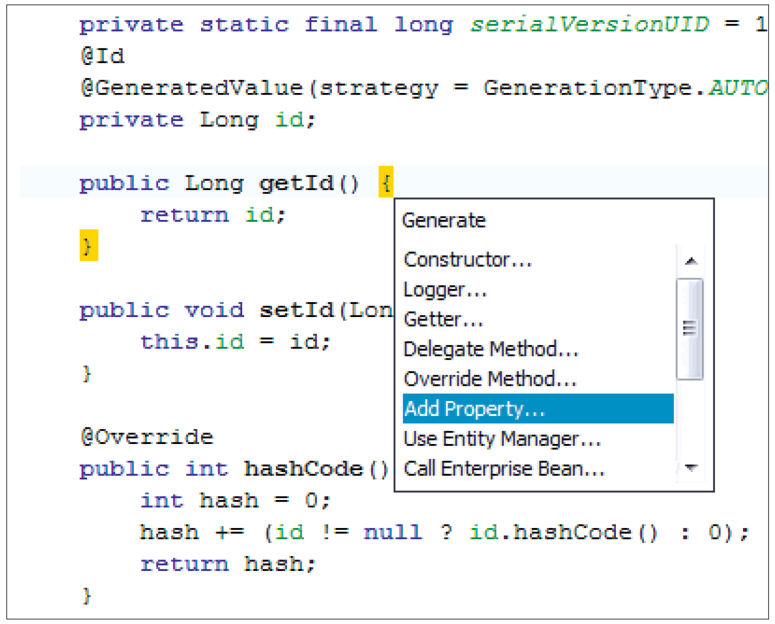
Persistence Unit Name: WebServiceClientSecureSSLPU
Specify the persistence provider and database for entity classes.
Persistence Provider: EclipseLink (JPA 2.0)(default)
Data Source:
☑ Use Java Transactio
Table Generation Strate

Figure 5

the datasource provided by NetBeans

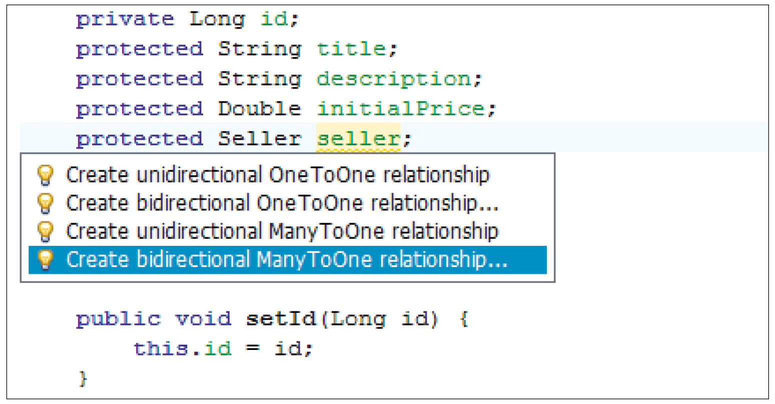
e. Click **Finish**.

3. Perform actions similar to Step 2 to create the Item and Bid entities.
Now we are going to add properties to the entities using the NetBeans wizard.
4. Open the Seller.java file, right-click anywhere in the code, and select **Insert code**.
5. From the Generate wizard, select **Add property**, as shown in **Figure 6**, and add the seller properties (String lastName, String firstName, and String email).
6. Open the Item.java file and add the item properties (String title, String description, Double initialPrice, and Seller seller).
7. To define the entity relationship, click the NetBeans warning lightbulb and select **Create bidirectional ManyToOne relationship**, as shown in **Figure 7**.
This action creates a list of items in the Seller entity.
8. Open the Bid.java file and add the item properties (String bidderName, Double amount, and Item item).
9. To define the entity relationship, click the NetBeans warning and select **Create bidirectional ManyToOne relationship**.
10. Generate the Getter and Setter, respectively, for the list of items and bids created in the Seller and item entities by right-clicking anywhere in the code and selecting **Insert code**.
11. From the Generate wizard, select **Getter and Setter**, as shown in **Figure 8**.
At this point, your Seller.java file will look like **Listing 1**.
12. Create the JavaServer Faces (JSF) pages:
The client implementation we are going to build consists of JSF pages based on the entities just created.
a. Right-click the **AuctionAppWebServiceClient** project and select **New**; then select **JSF Pages from Entity Classes**, click **Add all**, and click **Next**.
b. For **Session Bean Package**, type a name such as com.bonbhel.oracle.webServiceClientSecureSSL.facade and for **JSF Classes Package**, type a name such as com.bonbhel.oracle.webServiceClientSecureSSL.controller.
c. For **Folder Name**, type a name such as jsfClient.
d. Click **Finish**.
13. Create the Web service client:
We are going to use the Web Service Client wizard to create the Web service client. We will assume that the JAX-WS Web service is an external service that resides in the application tier over the network. So, we will use the URL to the JAX-WS Web service WSDL file.
a. Make sure the AuctionApp project is up and running. If it is not, right-click the **AuctionApp** node and choose **Deploy**.
b. Right-click the **WebService-ClientSecureSSL** node and choose **New**; then select **Web Service Client**.
c. In the **New Web Service Client** wizard, specify the URL to the Web service WSDL



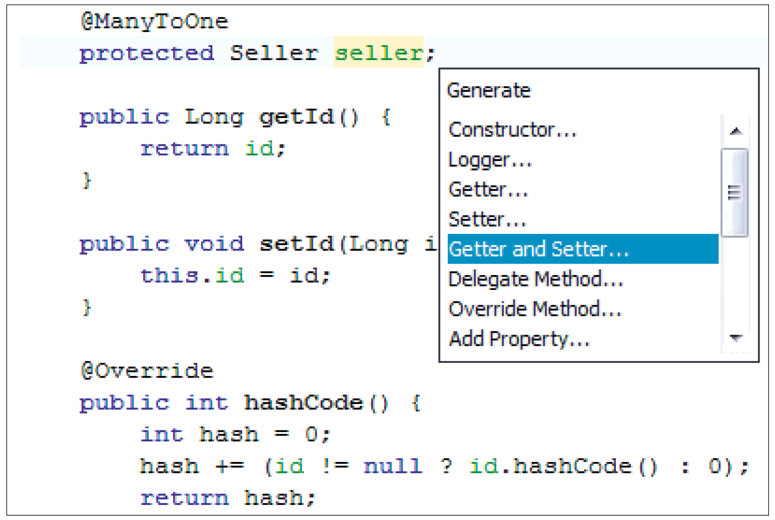
```
    private static final long serialVersionUID = 1
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO
    private Long id;

    public Long getId() {
        return id;
    }

    public void setId(Lon
        this.id = id;
    }

    @Override
    public int hashCode()
        int hash = 0;
        hash += (id != null ? id.hashCode() : 0);
        return hash;
    }
```

Figure 6



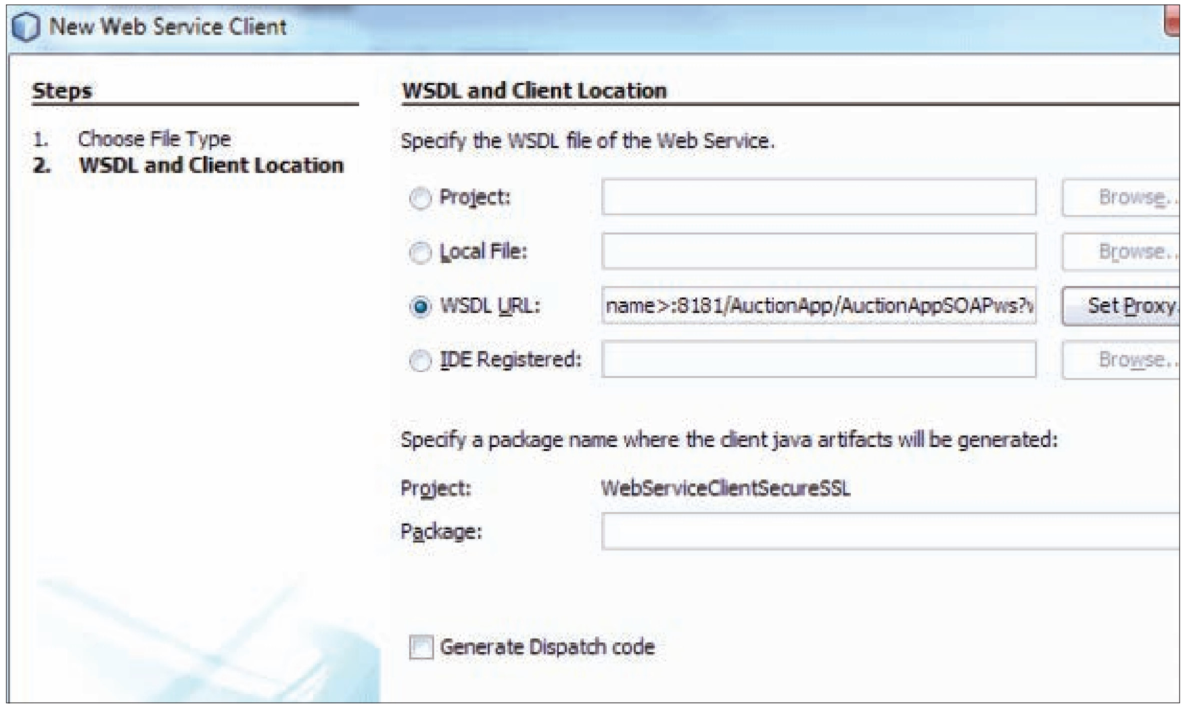
```
    private Long id;
    protected String title;
    protected String description;
    protected Double initialPrice;
    protected Seller seller;

    public void setId(Long id) {
        this.id = id;
    }
```
Create unidirectional OneToOne relationship
Create bidirectional OneToOne relationship...
Create unidirectional ManyToOne relationship
Create bidirectional ManyToOne relationship...

Figure 7



```
    @ManyToOne
    protected Seller seller;

    public Long getId() {
        return id;
    }

    public void setId(Long i
        this.id = id;
    }

    @Override
    public int hashCode() {
        int hash = 0;
        hash += (id != null ? id.hashCode() : 0);
        return hash;
```

Figure 8


New Web Service Client

Steps
1. Choose File Type
2. WSDL and Client Location

WSDL and Client Location

Specify the WSDL file of the Web Service.

Project:
Local File:
WSDL URL: name>:8181/AuctionApp/AuctionAppSOAPws?v
IDE Registered:

Specify a package name where the client java artifacts will be generated:

Project: WebServiceClientSecureSSL

Package:

Generate Dispatch code

Figure 9

file using the fully qualified host name, for example, https://<your-fully-qualified-hostname>:8181/AuctionApp/AuctionAppSOAPws?wsdl, as shown in **Figure 9**.

d. Accept all other default settings. The package name will be taken from the WSDL file.

e. Click **Finish**.

14. Add the extrapolateAmountBid operation provided by the JAX-WS Web service to the client classes:

a. Open the **Source Packages** node of the WebServiceClientSecureSSL project and double-click the BidController.java file located in the controller package com.bonbhel.oracle.auctionAppWebServiceClient.controller.

b. Put your cursor anywhere inside the source editor.

c. Expand the **Web Service References** node of the WebServiceClientSecureSSL project and drag the **extrapolateAmountBid** node inside the source editor. The extrapolateAmountBid() method appears at the end of the BidController class code, as shown in **Listing 2**. **Note:** Alternatively, you can right-click anywhere in the source editor and choose **Insert Code**; then select **Call Web Service Operation** and click the **extrapolateAmountBid** operation in the Select Operation to Invoke dialog box.

15. We need to add some application logic in the BidController class in order to extrapolate the amount of the bid (by factor, which is 100) when the user edits or views the bid entry. So call the extrapolateAmountBid operation to extrapolate the bid:

a. Open the BidController.java file in the source editor.

b. Modify the public String prepareView() method as shown in **Listing 3**.

At this point, the secured Web service client is regenerated and references the secured Web services WSDL file.

Test the Secured Web Service

Now it's time to test the service. We will invoke the secured Web service from the client application.

In this section, we are going to use the secured Web client application we created in the previous sec-

LISTING 1 LISTING 2 LISTING 3

```
@Entity
public class Seller implements Serializable {
  @OneToMany(mappedBy = "seller")
  private List<Item> items;
  private static final long serialVersionUID = 1L;
  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private Long id;
  protected String firstName;
  protected String lastName;
  protected String email;

  public List<Item> getItems() {
    return items;
  }

  public void setItems(List<Item> items) {
    this.items = items;
  }
```

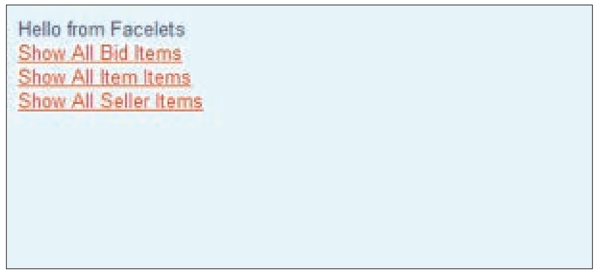
Download all listings in this issue as text

tion (WebServiceClientSecureSSL) and the unsecured Web client (AuctionAppWebServiceClient) to perform the following tasks:

- Try to invoke the secured Web service from the unsecured client application included in the project
- Use the secure client to invoke the secured Web service

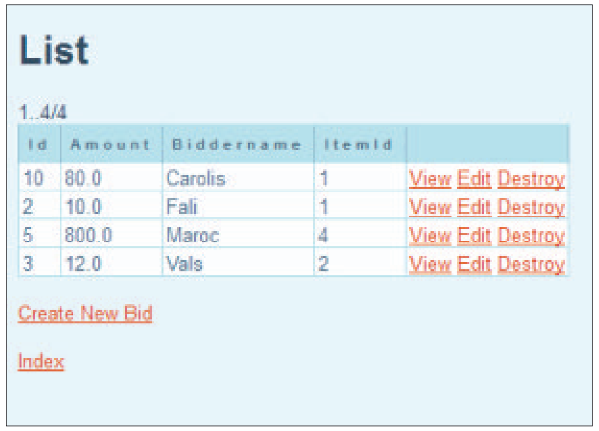
1. Try to invoke the secured Web service from the unsecured client to display the extrapolated amount of a bid:

   a. Make sure the AuctionApp project is up and running. If it is not, right-click the **AuctionApp** node and choose **Deploy**.



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

Figure 10



List

1..4/4

Id	Amount	Biddername	ItemId	
10	80.0	Carolis	1	View Edit Destroy
2	10.0	Fali	1	View Edit Destroy
5	800.0	Maroc	4	View Edit Destroy
3	12.0	Vals	2	View Edit Destroy

Create New Bid

Index

Figure 11

   b. Open the **Web Service** node of the AuctionAppWebServiceClient project and right-click the **AuctionAppSOAPws** node; then select **Refresh**.

   c. From the Confirm Client Refresh wizard, make sure **Also replace local wsdl file with original wsdl located at:** is selected; then click **Yes**.

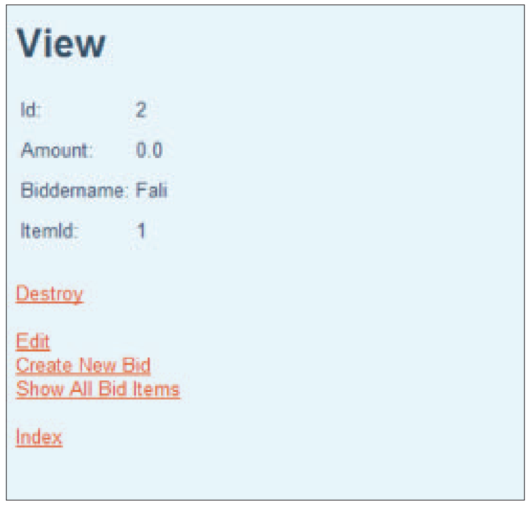
   d. Right-click the **AuctionAppWebServiceClient** project and choose **Clean and Build**.

   e. Right-click the **AuctionAppWebServiceClient** project again and choose **Run**. The list of all entries is displayed, as shown in **Figure 10**.

   f. Click the **Show all Bid Items** link to display the list of bid entries, as shown in **Figure 11**.

   g. Click the **View** link for the bidder named Fali to see the newly extrapolated amount of the Fali bid, as shown in **Figure 12**.
   As you can see, the amount of the bid changed from 10.0 to 0.0. This means that the client failed to call the secured Web service.

2. Use the secured Web client (WebServiceClientSecureSSL) to invoke the secured Web service to display the extrapolated amount of the bid:



View

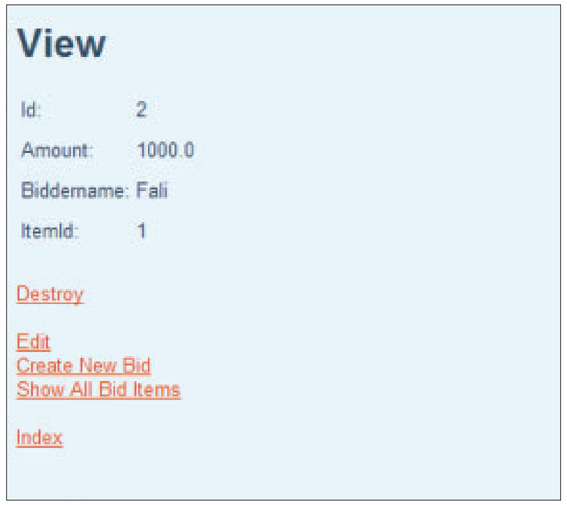
Id: 2
Amount: 0.0
Biddername: Fali
ItemId: 1

Destroy

Edit
Create New Bid
Show All Bid Items

Index

Figure 12



View

Id: 2
Amount: 1000.0
Biddername: Fali
ItemId: 1

Destroy

Edit
Create New Bid
Show All Bid Items

Index

Figure 13

   a. Make sure AuctionApp is up and running. If it is not, right-click the **AuctionApp** node and choose **Deploy**.

   b. Right-click the **WebServiceClientSecureSSL** project and choose **Clean and Build**.

   c. Right-click the **WebServiceClientSecureSSL** project again and choose **Run**.
   The list of all entries is displayed, as shown in **Figure 10**.

3. Click the **Show all Bid Items** link to display the list of bid entries, as shown in **Figure 11**.

4. Click the **View** link for the bidder named Fali to see the newly extrapolated amount of the Fali bid, as shown in **Figure 13**.
   As you can see, the amount of the bid changed from 10.0 to 1000.0. This means that the client was able to call the secured Web service. Excellent!

Conclusion

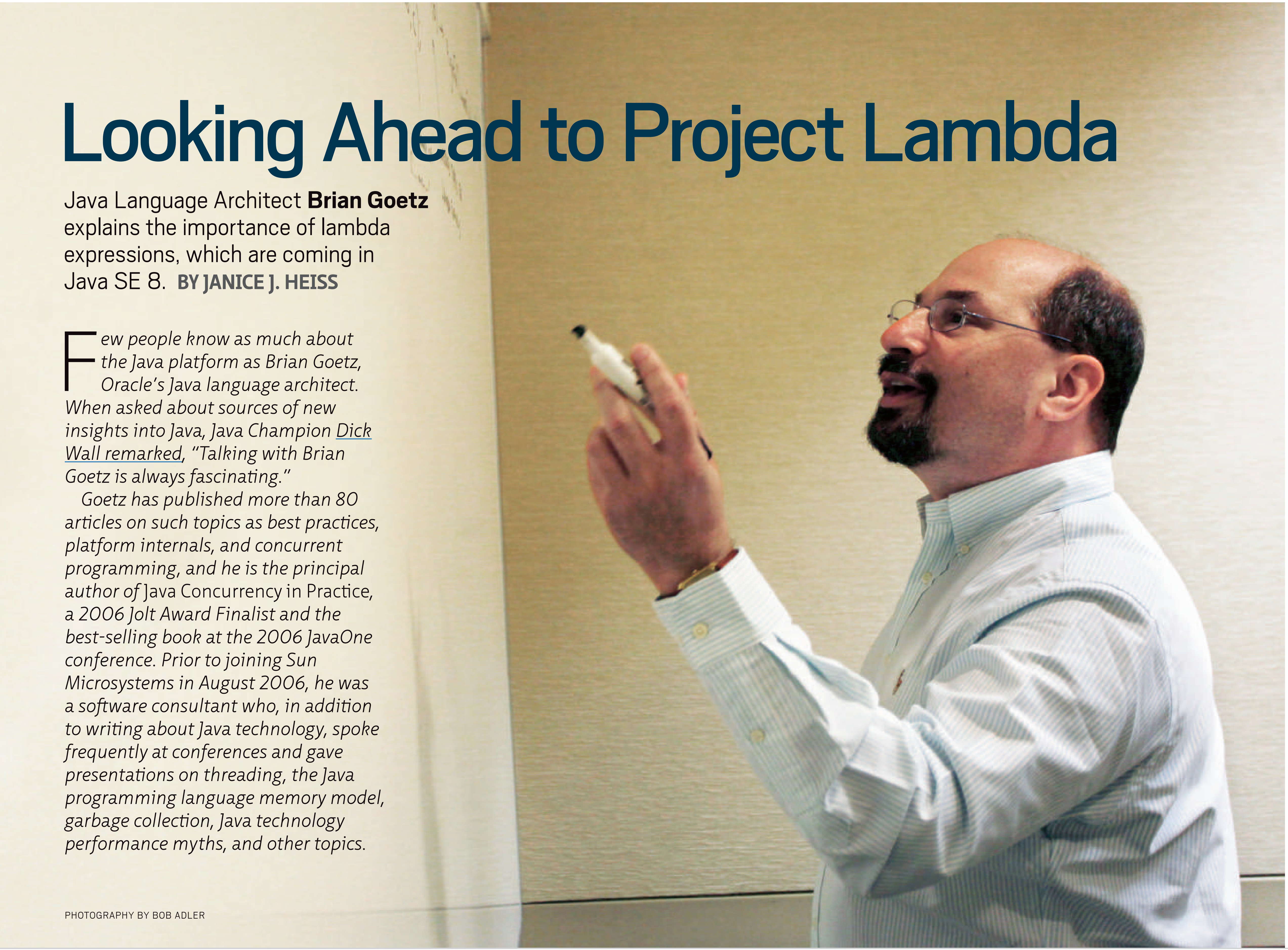
In this article, we have seen how easy it is to add transport-layer security to an existing application using SSL via HTTPS to protect the data transiting between the client and the server.

NetBeans and GlassFish make adding security for Web services easier than ever.

In the next article in this series, Part 3, we will focus on creating authorized users for the Web service. `</article>`

LEARN MORE

- NetBeans *Advanced Web Service Interoperability manual*
- *Metro User Guide*
- GlassFish resources



Looking Ahead to Project Lambda

Java Language Architect **Brian Goetz** explains the importance of lambda expressions, which are coming in Java SE 8. **BY JANICE J. HEISS**

Few people know as much about the Java platform as Brian Goetz, Oracle's Java language architect. When asked about sources of new insights into Java, Java Champion Dick Wall remarked, "Talking with Brian Goetz is always fascinating."

Goetz has published more than 80 articles on such topics as best practices, platform internals, and concurrent programming, and he is the principal author of Java Concurrency in Practice, *a 2006 Jolt Award Finalist and the best-selling book at the 2006 JavaOne conference. Prior to joining Sun Microsystems in August 2006, he was a software consultant who, in addition to writing about Java technology, spoke frequently at conferences and gave presentations on threading, the Java programming language memory model, garbage collection, Java technology performance myths, and other topics.*

PHOTOGRAPHY BY BOB ADLER

Java Language Architect Brian Goetz says a productive language is often one where the obvious solution is also a good solution.

Java Magazine: What will Lambda offer the "average" Java developer? Can you give us specific types of cases where developers will be able to do something because of Lambda that they would not otherwise do?

Goetz: In a trivial sense, since Java is already Turing-complete, additional language features don't expand the set of programs that can be expressed. But in practice the story is quite different. The set of features offered by a given language determines what programs can be expressed *easily and cleanly*. And this is important, because developers are human. A productive language is often one where the obvious or easy solution is also a good solution—because this keeps our natural laziness from leading us into harm.

The average developer's first experience with lambda expressions is likely to be through the new APIs for manipulating collections. Lambda is not just about language features but it is also about libraries, and together these constitute a significant upgrade to the programming model.

Adding lambda expressions—known also as *closures*—to the Java language provides developers, especially API designers, with a sharper tool for abstracting over behavior. While Java developers already have some tools with which to do so, the compactness of lambda expressions will definitely change the game in terms of what code is natural to write.

Take collections. The way to enumerate the elements of a collection is to ask for an iterator, and pull elements from the iterator, a task that is automated by the for-each loop:

```
for (Person p : people) {
  // do stuff with p
}
```

This is an idiom called *external iteration*. It is perfectly straightforward, but it has a lot of accidental characteristics that have nothing to do with the task of operating on each element of a collection—it is inherently sequential, elements must be processed in the order in which they appear in the collection, and the client is responsible for the mechanics of iteration. On the other hand, *internal iteration* lets the client pass some behavior to the collection to be applied to each element:

```
people.forEach(p -> { /* do stuff with p
*/ });
```

While this may seem no different, some very big things have changed—the library is now in control of the mechanics of the computation. While the client still specifies the *what*, it no longer specifies the *how*—and this is a good thing. Maybe the library will use parallelism, or exploit what it knows about memory locality to process elements in an order that is more efficient, or any number of other techniques for getting to the right answer faster. (Whether and when it should do these, and how they should be specified, are still hard questions, but in the internal iteration model they are at least possible, whereas in the external iteration model they are outright impossible.) Since the goal of writing libraries is to enable reuse of code written by experts, moving more details of *how* into the library opens a lot of doors for more expressive and performant libraries.

Java Magazine: How does this change the character of libraries?

Goetz: Lambda expressions enable the development of APIs where the library can retain control of a computation, but allow it to be more easily customized by the client; in turn, this generally requires the client to do less work to use them effectively. This affects

Working on Java SE 8's lambda expressions keeps Goetz on the run.

the APIs that we write, which in turn affects the character of the Java code that actually gets written. Another way to think of this is that having lambda expressions in the toolbox enables APIs to be more "permeable." External iteration provides a rigid demarcation between the library behavior and the client; the library hands up data one element at a time, and the client takes it from there. Internal iteration allows the client and library to cooperate in a more fine-grained way, each providing the portions of behavior they are best suited to provide.

For example, take the problem "find the groups that have at least one employee over 65, and print them in increasing order of size." Today, we might write this as follows:

```
List<Employee> people = ...
Set<Group> groups = new HashSet<>();
for (Person p : people) {
  if (p.getAge() >= 65)
    groups.add(p.getGroup());
}
List< Group> sorted = new
ArrayList<>(groups);
Collections.sort(sorted, new
Comparator<Group>() {
    public int compare(Group a, Group b)
{
        return Integer.compare(a.getSize(),
b.getSize());
    }
});
for (Group g : sorted)
  System.out.println(g.getName());
```

This is kind of ugly. We have several *garbage variables* here—groups and sorted—that exist only to hold intermediate results. The code is very ad hoc; small changes to the problem statement will have big ripple effects on the code. There's a lot of copying of data to get it into and out of the intermediate collections. And you have to read the data carefully to see what is going on.

Lambda-enabled collection libraries offer a much cleaner way to express this computation:

```
people.filter(p -> p.getAge() > 65)
    .map(p -> p.getGroup())
    .removeDuplicates()
    .sorted(comparing(g -> g.getSize())
    .forEach(g -> System.out.println(g.
getName());
```

There are no garbage variables, and you can read the computation from top to bottom and see exactly what is going on. It is the ability of the library to let the user parameterize the computation with little bits of functionality that drives this sort of library design, which makes things easier (and often more efficient) for the user. Moreover, small changes to the problem statement are more likely to result in small changes to the code.

Java Magazine: Which kinds of Java developers will most benefit from Lambda?

Goetz: Everybody will benefit—eventually. Library designers will benefit by having better tools for designing APIs; users will benefit by having richer libraries; organizations will benefit by having a greater ability to rely on powerful, performant libraries.

GAME CHANGER

The compactness of lambda expressions will definitely change the game in terms of what code is natural to write.

Java Magazine: Which kinds of developers, or developers faced with certain situations, will have to make the greatest adjustments because of Lambda? Will Lambda, under certain conditions, make life harder for some developers?

Goetz: Change always has a cost. This is not simply a matter of some new syntax; there are some new concepts here, and developers will have to learn them just to be able to read Java code, even if they do not plan to take advantage of these new features in the code they are writing. And there are also big additions to the core libraries; developers will have to learn these, too. But these costs should be more than made up for by increases in productivity and expressiveness.

Java Magazine: What does Project Lambda have to do with parallelism?

Goetz: The examples we've given so far are simply about structuring libraries to enable more expressive, less error-prone code, and this is certainly one of the key goals. But another key goal is lowering the bar to exploiting hardware parallelism. While we added the fork/join framework in Java SE 7, which helps developers write algorithms that can run efficiently on a wide range of core counts, there is still a big gulf between the source code for serial and parallel algorithms. If I wrote the code for doing the "groups with employees over 65" query to run in parallel using fork/join, you'd hardly recognize the code. It would fill a page and would look almost nothing like the sequential implementation for that problem.

By migrating from an external iteration model to an internal iteration model, parallelism is one more thing that we can move into the library. The above query can be parallelized with a small change:

```
 people.parallel()
   .filter(p -> p.getAge() > 65)
   .map(p -> p.getGroup())
   .removeDuplicates()
   .sorted(comparing(g -> g.getSize())
   .sequential()
   .forEach(g -> System.out.println(g.
getName());
```

All that has changed is the insertion of the parallel() and sequential() calls; the filter/map/sort can now happen in parallel, and at the end when we want to process the elements in order, we switch back to serial mode. Now, obviously we have to do some work to add parallelism to the libraries, but the internal iteration model makes this at least possible.

Java Magazine: How does Lambda address the problem of the aging of Java libraries?
Goetz: The Java Collections framework was added in Java 1.2, and the core abstractions—Set, List, and Map—have remained unchanged for almost fifteen years. This is because once published, we cannot add new functionality to an interface without breaking existing implementations. Ironically, adding lambda expressions to the Java language makes this problem worse. Once we have lambda expressions, we immediately begin wanting additional methods, such as Collection.forEach. (Whenever you add a language feature, you put pressure on existing APIs, since some of them would likely have been written differently had the feature been available at the time they were written.)

Because adding lambda expressions to Java had the unintended consequence of making our aging Collections APIs look even older, we decided to pair lambda expressions with another feature—default methods. This feature allows you to compatibly add a new method to an interface if the interface provides a *default implementation*—a fallback in case classes implementing the interface don't provide one. These default methods are fully virtual methods, like any other interface method—classes can override them or not, as they please. This is how we're going to add the new methods to existing APIs to take advantage of lambda expressions.

This feature bears similarities to features from other languages, but this is a slightly different approach. C# has extension methods, but its extension methods are static, whereas Java's default methods are virtual. C++ has unrestricted multiple inheritance. Default methods allow for multiple inheritance of behavior (Java always had multiple inheritance of types), but not multiple inheritance of state (which is where many problems in C++ inheritance come from). Scala has traits, but default methods are more limited than traits. These differences are motivated by our goals—our primary goal in adding default methods was *interface evolution*.
Java Magazine: Two concepts jump out when I look at examples of the new Lambda-enabled collections—*laziness* and *streams*. What is laziness? What are streams?
Goetz: Laziness has to do with the timing of when an operation occurs. Take the code fragment:

```
  bobs = people.filter(p ->
p.getFirstName().equals("Bob"));
```

When does the filtering actually occur?

IT'S OK TO SLACK
Laziness can be a big performance advantage. If we were doing the filtering eagerly, we'd have to evaluate the predicate on every element of the input.

There are two choices: All the filtering is complete on return from the filter() method, or the filtering happens as we attempt to pull elements from bobs. The former is termed *eager*; the latter is termed *lazy*. The choice of whether to compute eagerly or lazily is another one of those implementation flexibility choices that libraries acquire under an internal iteration model.

Laziness can be a big performance advantage. If we were doing the filtering eagerly, we'd have to evaluate the predicate on every element of the input. But what if we wanted only the first result, and the list had a million elements? We'd have wasted a lot of time. (Laziness also opens the doors to calculations on infinite data sets.) In order to achieve laziness, the filter() method returns not a fully populated collection but a stream. A stream is like an iterator; you can pull values from it until it is empty, but it doesn't do any filtering until you actually ask it for the next element. By modeling calculations as streams, we can avoid having to stuff the data into intermediate collections as we do filtering, mapping, and so on and instead operate on the values as they flow by. This allows queries like the ones above to be not only more readable but more efficient, too. In the sequential example of the "groups with employees over 65" query, we don't have to populate intermediate collections that are about to be thrown out. The parallel version can actually be done in a single parallel pass, rather than separate passes for filtering, mapping, and sorting. Laziness is what enables these characteristics.

Java Magazine: What kinds of feedback would you like from the Java community about Lambda and Java SE 8 in its current stage of development? What questions would you like them to answer, and how can they check out the code?

Goetz: The single most important thing that the community can do to help is try it out. Download binaries and try using the new Collections features or using lambda expressions in your own code, and report your experiences.

TAKE CARE

> Careful programmers will find that lambda expressions can make their code clearer, cleaner, and easier to read; careless programmers will no doubt use them to create code that is even more cryptic, poorly factored, and confusing.

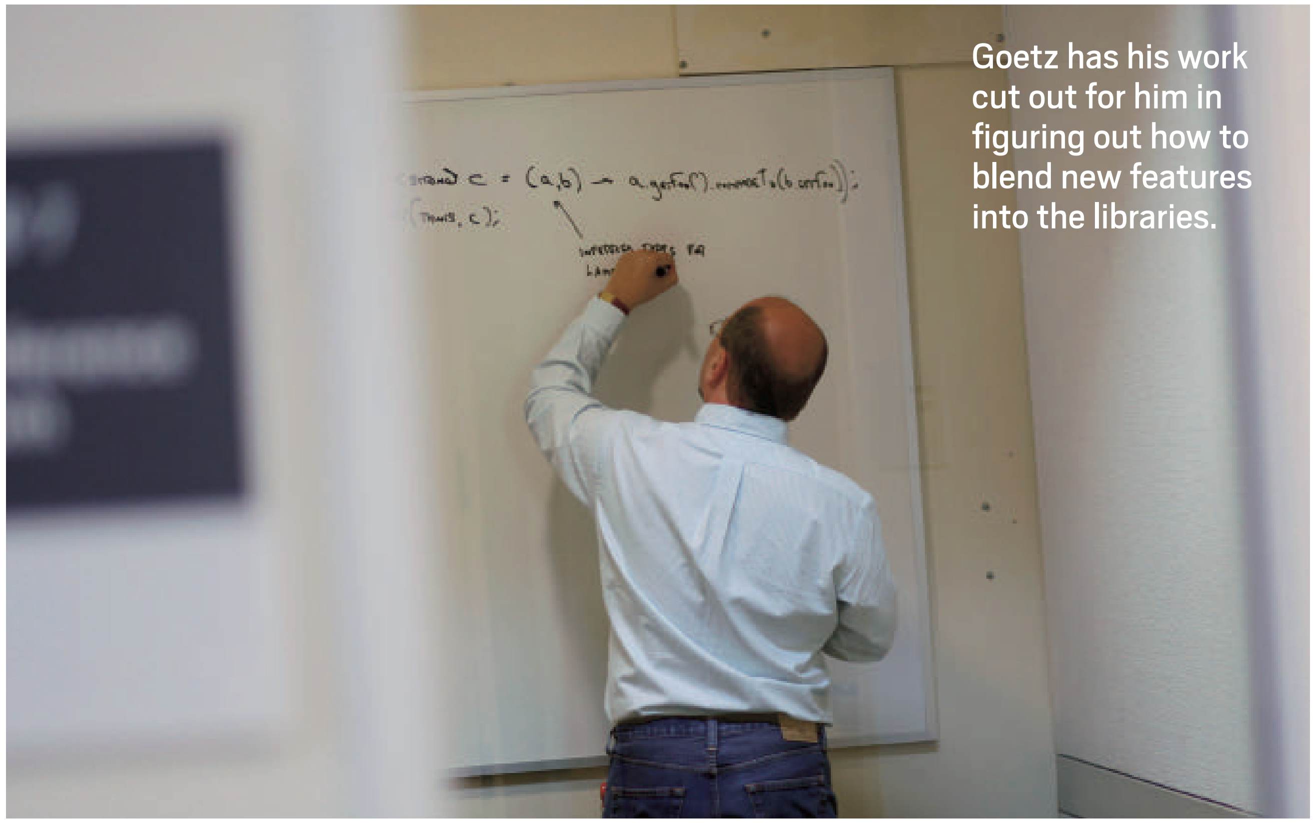
Java Magazine: Some developers appear to be afraid that lambda expressions might be very clever but also very difficult to understand and maintain by someone who is not the author. Can you address this concern?

Goetz: This is a risk, but it's a risk that is present without lambda expressions, too. It is possible to write very clear, clean, easy-to-read code in Java, and it is also possible to write cryptic, poorly factored, confusing code. Careful programmers will find that lambda expressions can make their code clearer, cleaner, and easier to read; careless programmers will no doubt use them to create code that is even more cryptic, poorly factored, and confusing.

One of the hidden dividends of the changes coming in JDK 8 is a gentle push away from mutability. In the example above, the "old" version did everything via mutation—populating and modifying collections of intermediate results. The second version is done without any mutation by the client, and this ends up being less error-prone. This also works better with the hardware trend toward multicore systems.

Java Magazine: What open questions in Lambda still have to be dealt with?

Goetz: There's still a lot of work left to do with the libraries. The examples above are illustrative, but we're still working on exactly how these features will be blended into the libraries. </article>


Goetz has his work cut out for him in figuring out how to blend new features into the libraries.

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

LEARN MORE

- Project Lambda
- Download Lambda binaries
- JSR 335: Lambda Expressions for the Java Programming Language
- Brian Goetz' blog

BEN EVANS AND
PETER LAWREY

BIO

BEN EVANS' PHOTOGRAPH BY JOHN BLYTHE

Part 2

Inside the Java HotSpot VM 2: Statistics for Performance Analysis

Why benchmarking at the macro level, understanding the distribution shape, and measuring from the client's perspective are critical.

In the first article in this series, we introduced the Java HotSpot VM and some of the basic concepts related to runtime compilation of methods. If you haven't already read that article, you may want to check it out before continuing.

In this article, we want to discuss some of the subtleties that are inherent to performance analysis on the Java platform. Specifically, we want to look at the handling of statistics for observed method timings—especially microbenchmarks.

There are many blog posts and articles on the internet that draw wrong conclusions about Java performance, because they do not take into account the statistical behavior of Java performance numbers at the micro level.

In writing this article, we hope to show the difficulties involved in accurate microbenchmarking and to encourage the reader to focus on benchmarking at the macro level—that is, at the level of whole applications and application tiers—rather than focusing on small methods and very short sections of code.

Let's start by looking at a modified version of the code we used in the last article (see **Listing 1**).

This code is essentially the same as the code we used last time to show the effect of just-in-time (JIT) compilation and inlining on simple methods, such as getters and setters, but we've added two features:

- The ability to run with a small amount of object allocation on each iteration. This will cause garbage collection to eventually run.
- Instrumentation to allow us to collect the length of each individual iteration.

In the last article, we just calculated the averages and didn't worry about what the actual timings were for each run. However, we need a better level of detail to fully explore the statistical aspects we want to investigate. We'll use the class in **Listing 2** to manage the runs.

Let's look at some output for a run without allocation on a 2010 MacBook Pro using Java SE 7u4 (see **Listing 3**).

NOT THE NORM

Typical values for performance metrics obtained from a JVM do not usually form a normal distribution.

The calculated average for the actual run is 44.9 ns, but the 50th percentile is 0, which would imply that the code took no time at all to run. These results seem to contradict each other.

In fact, this effect is caused by the Mac OS X timing subsystem, which records time only at the microsecond level, which causes System.nanoTime() to return results that are whole thousands of nanoseconds (because 1,000 ns equals 1 mu). This shows how important it is to have an awareness of the hardware and OS features that can affect the performance numbers produced on a specific platform.

On other operating systems and hardware, the granularity of nanoTime() is actually nanoseconds. From here on, we'll use the same hardware and Java Virtual Machine (JVM), but we'll run under Ubuntu Linux, so we can obtain the finer precision we need for individual runs.

The Distribution of Results

One very important aspect of gathering good numbers for performance tuning is to understand the distribution of performance-sensitive results. In particular, typical values for performance metrics obtained from a JVM do not usually form a normal distribution.

Instead, a performance metric, such as the time taken to execute a method, is typically composed of two parts: the underlying result plus some noise introduced by the JVM. In general, the amount of noise introduced by the JVM is always positive, and it might be much larger than the value being measured.

This means that the shape of the distribution is not a normal (also called a *Gaussian* or *bell curve*) distribution, but instead it will be closer in shape to a distribution with an asymmetric, fat tail. In overall shape, it will resemble a Gamma distribution, as shown in **Figure 1**, but it will have individual outlying points at large values.

In order to see how this occurs, let's consider the effect of one of the most important components of the JVM: garbage collection (GC). As we know, the JVM mark-and-sweep algorithm causes all application threads to pause periodically. This means that some instrumented methods might be running when GC begins. The results will then appear to show those runs as being much slower than normal, because the measured method is paused while GC is

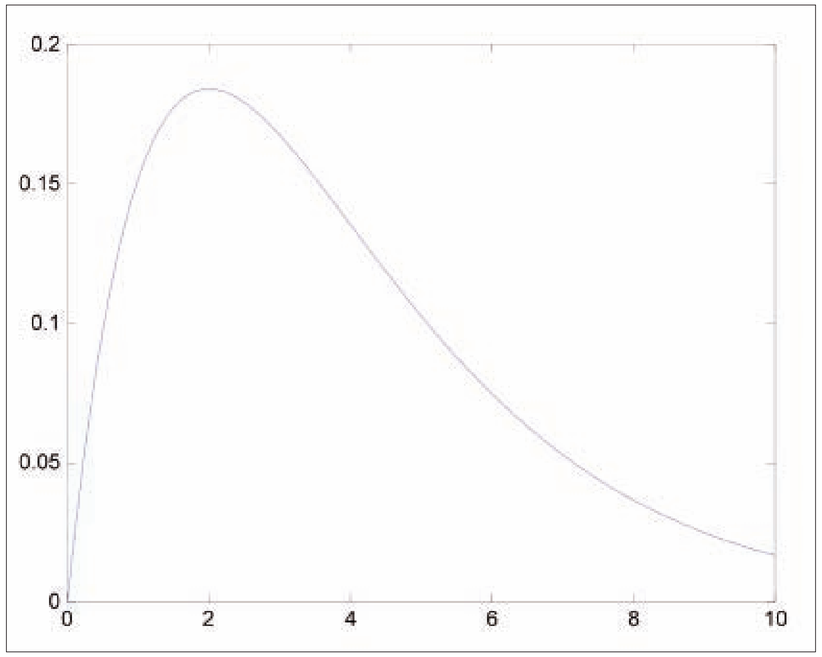


Figure 1

LISTING 1 / LISTING 2 / LISTING 3

```
import java.util.UUID;
import java.util.concurrent.Callable;

public class GetSetCallerWithAlloc implements Callable<Double> {
  private final int runs;
  private final long[] results;
  private final boolean doAlloc;

  public GetSetCallerWithAlloc(long[] res, boolean alloc) {
    results = res;
    runs = res.length;
    doAlloc = alloc;
  }

  @Override
  public Double call() {
    ViaGetSet getSet = new ViaGetSet();
    double sum = 0;
    UUID uuid;
    long end, start = System.nanoTime();
    for (int i = 0; i < runs; i++) {
      getSet.setOne(getSet.getOne() + 1);
      sum += getSet.getOne();
      if (doAlloc) {
          uuid = UUID.randomUUID();
          sum += uuid.toString().length();
      }
      end = System.nanoTime();
      results[i] = end - start;
      start = end;
    }
    return sum;
  }
}
```

Download all listings in this issue as text

running. The effects of GC can be quite severe.

To demonstrate these effects, let's look at some results that show two runs—one without allocation and one with—to show the extent of the problem of outlying points. To fully illustrate the issue, we'll show the time taken to reach certain percentile levels. To read these results, recall that the 99 percent level is the duration that 99 percent of all timings were less than.

Listing 4 is from Ubuntu Linux 11.04 on a MacBook Pro 2010 with JDK 7u4 and no allocation, and **Listing 5** is from Ubuntu Linux 11.04 on a MacBook Pro 2010 with JDK 7u4 and allocation.

There are several things to notice. The first is that even with no allocation, there are still a few results that are outliers. These are caused by effects such as the action of the operating system scheduler.

ONE CAVEAT

Too often, tools such as the standard deviation are applied without reference to the caveats that apply to their use.

The most pronounced effect, though, is with the addition of allocation. This causes the 90th percentile value to move from 23 ns to 5.00 mu—that is two orders of magnitude. The introduction of allocation caused the effect that we're trying to measure to get completely swamped. Such are the perils of writing microbenchmarks on the JVM; these really are very specialized cases that should be done only by experts who are used to working with the JVM at a low level.

Rather than trying to benchmark extremely small sections of code and deduce upward, most developers should focus on larger-scale observables that are directly relevant to an application's business goals, such as overall throughput or end-to-end response time.

LISTING 4 LISTING 5

```
  71 1,000,000          field, getter/setter=33.1 ns
 110 1,000,000          field, getter/setter=25.1 ns
50.0% delay was 23 ns
90.0% delay was 30 ns
99.0% delay was 43 ns
99.9% delay was 164 ns
99.99% delay was 248 ns
99.999% delay was 3,458 ns
99.9999% delay was 17,463 ns
```

Download all listings in this issue as text

Statistical Rigor

The nature of the distribution of Java microperformance results also leads to a need for caution when dealing with standard statistical measures. In particular, measures such as the standard deviation can become very unreliable when dealing with these types of results.

Too often, tools such as the standard deviation are applied without reference to the caveats that apply to their use. Many people remember the "rule" that "68 percent of values lie within one standard deviation of the mean, and 95 percent lie within two standard deviations" without remembering that in deriving that rule, the assumption is made that the distribution is normal.

If the distribution is not normal, there is no general rule that can be made about the percentiles of an arbitrary probability distribution.

In order for our handling of statistics to be watertight, we need to do more than account for the nonnormal distribution of our results. In particular, we need to ensure that we perform enough independent runs to guarantee that our results are not affected by one run that might be heavily skewed by external noise (other processes, operating system tasks, or other system effects).

The results that we presented previously were chosen from a large set of independent runs and were checked to ensure that they were truly representative of the overall statistics.

The benchmark paper on the proper handling of statistical results from the JVM is "Statistically Rigorous Java Performance Evaluation" by Georges, Buytaert, and Eeckhout (2007), which is available from several sources on the Web. For applications that need a very high level of rigor and certainty, the methods described in that paper should be followed closely.

Finally, there is another very important subtlety of measurement that we have so far neglected.

The Client View

Consider a system, such as a Web server, for which we want to measure response times. If we follow the measurement approach that we've seen so far, we would conclude that a certain small percentage of the server's page response times would be slow—the pages that were unlucky enough to be in-flight when a GC pause started.

However, the situation is actually worse than that. Considering the time from the server's point of view is not sufficient. This is because requests continue to be sent by clients even though the JVM is paused for GC. These requests will begin to be processed after the GC cycle has completed, and so the measurements of their duration will not show any sign of having been held up by GC.

From the point of view of the sending client, however, these requests will certainly have been affected by GC, and they will seem slow to the client.

For this reason, a truly accurate view of a performance number, such as response time, must take into account the complete timings as seen by the client as well as the character of the distribution of results.

Summary

In this article, we've discussed the nature of JVM microbenchmarks and shown some of the pitfalls of developing good microbenchmarks:

- The dynamic, managed nature of the JVM affects performance numbers.
- GC can have an especially large effect.
- Microbenchmark observables are not normally distributed.
- Simple descriptive measures (such as the mean or standard deviation) can be very misleading for microbenchmarks.

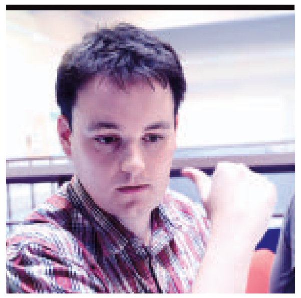
Most application developers should look to benchmark at a much larger scale by working with their entire application or a component. Even in this case, there are a number of basic benchmarking principles that developers should follow when benchmarking:

- Understanding the overall shape of the distribution of performance indicators is important; don't rely on statistical measures without checking the overall shape of the distribution first.
- Make sure that enough independent runs are done to achieve statistical stability.
- Beware of the "queuing on the network" effect. To be completely accurate, we must measure from the client's perspective. </article>

LEARN MORE

• Java HotSpot VM

MAKE THE FUTURE JAVA
Java™
FIND YOUR JUG HERE
I have met so many amazing people through the London Java Community—friends, mentors, new colleagues—and through it I have achieved much more than I could have on my own.
Trisha Gee
London Java Community (LJC)
LEARN MORE
ORACLE®



New File System APIs in NIO.2: A Walkthrough

Take advantage of the new file system APIs released in Java SE 7.

JULIEN PONGE

BIO

PHOTOGRAPH BY
MATT BOSTOCK/GETTY IMAGES

Java SE 7 provides new additions to the NIO packages—the so-called NIO.2 APIs of JSR 203—most of which focus on file system APIs and asynchronous channels.

In the past, most file system operations in Java were performed using the java.io package classes that provide an abstraction of system-dependent features. These classes are weak, for example, in dealing with file permissions and handling symbolic links. Many libraries were developed to overcome these weaknesses, but they have the added expense of relying on either native libraries or external process invocations. Other libraries were also developed to better deal with common file system operations (such as copy, move, and exploration) or to provide virtual file systems (such as compressed archives, FTP, and HTTP).

WEAK LINK
In the past, most file system operations in Java were performed using the java.io package classes, **which are weak** in dealing with file permissions and handling symbolic links.

In this article, we focus on the NIO.2 additions that are related to file systems: operations, exploration, metadata manipulation, and change monitoring. As we will see, the NIO.2 APIs provide elegant additions to the Java toolset.

First Steps

The NIO.2 additions are mostly contained in the java.nio.file packages. Let's get started with Path, a key interface of the new APIs.

Paths and files. The Path interface is an abstraction of a file or a directory within a given file system. Instances of Path can be obtained using the Paths class, which comprises two get() methods: one taking a URI and one taking a variable number of path elements. An example would be to get a Path reference to the /etc/hosts file, as shown in **Listing 1**.

A Path reference offers methods for comparing paths, resolving paths against other paths, and so on. The Files class is commonly used to perform common operations on paths, including the following:

- Creating files, directories, and symbolic links
- Copying, moving, and deleting
- Querying for attributes
- Iterating over a file system tree
- Obtaining read and write streams and channels
- Performing direct read and write operations

Listing every method in Path and Files would certainly be boring, so let's instead look at some examples.

Simple operations. Let's create a temporary directory and copy a file into it.

Listing 2 copies the asm file to an asm-copy.jar file under a temporary directory such as /var/folders/cj/h0x5ghcd1tz5s7shyj8txxf0000gn/T/nio2132476771714525O358.

Several or no CopyOption parameters can be provided, most likely from the StandardCopyOption enumeration. In **Listing 2**, we asked to preserve the file system attributes, such as the owner, group, and permissions. Other options include the ability to perform an atomic move and replace existing files. There is also a LinkOption enumeration to prevent the copy operation from following symbolic links.

Speaking of symbolic links, if a symbolic link to the original ASM library Java archive (JAR) didn't exist, we could create one and query the size of the JAR, as shown in **Listing 3**.

Quick reads and writes. Reading and writing file contents has traditionally required lots of code. The Files class provides quick helper methods for reading and writing the contents of files as one-shot operations. They operate either on strings or raw byte arrays.

Listing 4 shows how we can read the contents of a file. Writing some content to a file is equally easy, as shown in **Listing 5**.

A PRICE TO PAY

Many libraries were developed to overcome weaknesses in the java.io package classes, but they have the added expense of relying on either native libraries or external process invocations.

Integration with existing stream and NIO APIs. NIO.2 APIs are not intended to live on an isolated island, so they provide a means for integration with existing stream and NIO APIs.

I/O java.io style streams and buffers can be obtained using the Files class and its newInputStream(), newOutputStream(), newBufferedReader(), and newBufferedWriter() methods. NIO channels can be obtained using the newByteChannel() method.

Interestingly, there is a version of the copy() method that can be used when the target is an instance of java.io.OutputStream.

An example usage would be to dump the contents of a file to the standard output stream:

```
Files.copy(hello, System.out);
```

Dealing with File Systems

As the name suggests, the FileSystem class represents an abstraction of a file system. It provides many useful services, including the ability to access files, directories, users, groups, or file system change notifications.

Obtaining and using a file system reference. FileSystem is an abstract class. Instances can be obtained using the corresponding FileSystems factory object.

FileSystem contains a getDefault() method that returns the *default* file system, that is, the file system from which the Java Virtual Machine was invoked. We can obtain a reference to this default file system like this:

```
FileSystem fs = FileSystems.
getDefault();
```

Not every file system shares the same properties. While UNIX-derivative operating systems share a unified file system root (/), Microsoft Windows has one root per partition and drive (C:\, D:\, and so on). We can check how many roots a given file system has by invoking the getRootDirectories() method, which returns an Iterable<Path>, as shown in **Listing 6**.

An instance of FileSystem can also be used to retrieve Path instances. The methods in Path include manipulation methods for comparing paths, extracting subpaths, computing relative paths between paths, and so on.

Obtaining instances of Path from a FileSystem object is done by invoking the getPath() method, which takes a variable number of string arguments. Each string represents a part of a path, as shown in **Listing 7**.

We can compare paths like this:

```
assert main.compareTo(alsoMain)
== 0;
```

compareTo() returns the usual -1, 0, and 1 values for "less than," "equal to," and "more than," respectively.

Listing 8 shows how a path can be converted to a URI or file representation. We can also perform some path manipulations, as shown in **Listing 9**.

Listing the file system providers. There are other factory methods in FileSystems that can build FileSystem instances for a URI or

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

```
// These two calls are equivalent
Path hosts1 = Paths.get("/etc/hosts");
Path hosts2 = Paths.get("/etc", "hosts");
// (...)
```

Download all listings in this issue as text

path in a file system. This is especially useful when concrete providers act as virtual file systems over zip archives, HTTP or FTP connections, or SMB network drives. Using this capability, you can uniformly handle "real" and virtual file systems.

It is possible to query the available file system providers by using the FileSystemProvider class, a service provider interface (SPI) that is part of the java.nio.file.spi package.

The code shown in **Listing 10** should return both file and JAR URI schemes for most installations of Java SE 7. Although the specification does not require a JAR scheme provider to be available, JDK 7 and OpenJDK provide one. You can take advantage of the SPI's ability to contribute its very own file system providers (HTTP, FTP, and so on), although doing that is beyond the scope of this article.

Let's now further investigate the NIO.2 APIs by opening a zip file as a file system provider and "walking it."

CLASS ACT
The NIO.2 APIs provide elegant additions to the Java toolset.

Walking a File System Provider

The java.io.File class has list() and listFiles() methods. They can be used for instances of File that represent a directory, and they return a list of direct children files and directories.

Slightly more-powerful mechanisms are provided with the java.nio.file.Files class: file visitors and directory streams, which can perform selective filtering and matching for discovered elements.

We'll look at those capabilities in the remainder of this section.

Opening a zip file system. Let's open a JAR as a file system, as shown in **Listing 11**.

The FileSystems class provides other newFileSystem factory methods, including one that takes a configuration map as a parameter and uses provider-specific keys. The format in **Listing 11**, which takes a Path and a ClassLoader, is the preferred format for opening a file system over a file.

Because FileSystem implements AutoCloseable, we can use it as part of a try-with-resources statement that ensures the FileSystem.close() method is called, regardless of whether an exception is thrown.

Path matchers. In this example, we are interested in printing out all the .class files contained within the archive. We can take advantage of PathMatcher to match Path objects against either a regular expression or file glob syntax using the match() method, as shown in **Listing 12**.

A path matcher expression has the form "glob:<glob-expression>" or "regexp:<regular-expression>". An expression such as "glob:/foo/*.{java,class}" matches any .java or .class file within /foo.

File visitors. The Visitor design pattern fits well with the need to walk a file system tree and take action on files. The FileVisitor<T> interface defines several methods for the following circumstances:

- A visitor that is notified before and after entering a directory
- A visitor that is notified when visiting a file
- A visitor that is notified when visiting a file through an exception

Also, file visitor methods return a value as part of the FileVisitResult enumeration to mandate that the walk be continued, be terminated, skip the siblings, or skip a subtree. There is also a default implementation called SimpleFileVisitor<T> that can be overridden. In our case, we can walk the archive file system from its root and print out .class file paths, as shown in **Listing 13**.

We can safely ignore the attrs parameter of the visitFile() method for the moment; we will examine how file attributes can be manipulated later.

Directory streams. File visitors are fairly customizable, but when all you need to look for is certain files in a single directory, a simpler solution exists in the form of directory streams. In a nutshell, a stream is simply an *iterable* object that runs over a set of paths that match a certain filter. It is also an AutoCloseable interface for use in

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

```
Path main     = fs.getPath("src", "discover", "nio2", "fs", "Main.java");
Path alsoMain = fs.getPath("src/discover/nio2", "fs", "Main.java");
Path src      = fs.getPath("src");
```

Download all listings in this issue as text

try-with-resources statements, because directory streams need to be closed.

Listing 14 shows how we can look for all entries at the root of the previous JAR, and **Listing 15** shows how we can pass a glob filter to list only the Java class files.

More-elaborate queries can be implemented by passing a DirectoryStream.Filter object as a stream predicate. For example, we can reuse the glob matcher object that we used earlier to list all Java classes, as shown in **Listing 16**.

As in the case of file visitors, the predicate does not need to be based on querying a PathMatcher. You could, for instance, look at file sizes or file attributes to form queries such as "any executable file over 1024kilo-octet in size."

Copying a whole file tree into a zip archive. We can easily combine a file tree walker, the Files class operations, and an ad hoc file system to copy the entire current directory file tree into a UTF-8 encoded zip archive, as shown in **Listing 17**.

If, instead of copying files into a zip archive, we had an FTP file system to copy files to a remote FTP server, the code in **Listing 17** would not change much, except for the file system creation. The Files .createDirectories() method covers the creation of all directories that need to exist in order to lead to the directory to be created.

SOMETHING COOL

One of the most interesting features of the NIO.2 API is the ability to manipulate metadata at the file system level.

Manipulating Metadata

One of the most interesting features of the NIO.2 API is the ability to manipulate metadata at the file system level. Previously, information such as file owner, group, and access permissions was not available from the standard Java runtime without resorting to external process execution or native code integration, both of which add their own sets of problems. While such information is no longer hidden, code needs to adapt to the capabilities of the runtime environment, because operating systems and file systems vary greatly.

The java.nio.file .attribute package provides support for accessing such metadata, mostly by using a hierarchy of *file attribute views*. Each view represents a set of metadata that can be manipulated depending on what the underlying file system provides.

The BasicFileAttributeView interface defines the attributes that are common to existing file systems, such as file size, last modification date, and creation date. Instances of DosFileAttributeView define attributes of DOS-type file systems,

LISTING 13 / LISTING 14 / LISTING 15 / LISTING 16 / LISTING 17 / LISTING 18

```
Files.walkFileTree(fs.getPath("/"), new SimpleFileVisitor<Path>() {
 @Override
 public FileVisitResult visitFile(Path file, BasicFileAttributes attrs)
   throws IOException {
  if (matcher.matches(file)) {
   System.out.println(file);
  }
  return FileVisitResult.CONTINUE;
 }
});
```

Download all listings in this issue as text

such as whether a file is hidden, a system file, or an archive. In the same spirit, PosixFileAttributeView defines attributes for POSIX systems, while AclFileAttributeView allows for accessing access control lists (ACLs) on file systems that support them.

The Javadoc for the java.nio.file.attribute package thoroughly describes these views. Let's look at some common usage patterns.

File attributes. The most-common file attributes are directly accessible from the Files class methods: size(), isDirectory(), isRegularFile(), isSymbolicLink(), isHidden(), getLastModifiedTime(), and more.

To access or modify a single attribute, you can use the getAttribute() and setAttribute() methods. They take a string that represents the attribute, prefixed with an identifier of the file attribute view. Each file attribute view interface defines such strings and the return type of the attribute in the corresponding Javadoc.

FIND IT HERE

The NIO.2 additions are mostly contained in the java.nio.file packages.

We can query the POSIX permissions for a file as shown in **Listing 18**, which would result in output similar to **Listing 19**. In addition, we can access specific attributes using the readAttributes() method on any subclass of BasicFileAttributes, as shown in **Listing 20**.

It is worth noting that Files.readAttributes(Path,BasicFileAttributes,LinkOption[]) performs bulk read operations. This capability is useful performance-wise when multiple attributes are read simultaneously.

POSIX file permissions can be directly accessed as shown in **Listing 21**. More-general cases can be handled by first accessing a file attribute view. For example, to obtain the POSIX file owner of a file, you can use the code in **Listing 22**.

POSIX file permissions are commonly expressed in the form of a string such as rwxr-xr-x, which means the user can read, write, and execute, while group members and others can only read and execute. We can display such permissions and change them, as shown in **Listing 23**, which produces output similar to this:

```
Old permissions: rw-r--r--
New permissions: rwxr-xr-x
```

The FileStore class. Every path has an associated *file store* object that essentially represents where the corresponding file is stored: volume, partition, and so on. The FileStore class is useful for querying whether file storage supports a given file attribute view. It is also useful for accessing disk usage statistics.

The code in **Listing 24** allows us to perform a few queries that produce output similar to the following on Mac OS X:

```
/dev/disk0s2
hfs
169030116 / 244277768
true
true
false
```

Watching for Changes

Some applications need to watch for file system changes. For example, an application server might watch for application archives to be dropped into a certain folder and then automatically deploy them. Similarly, integrated development environments can look for project files to be added, deleted, and modified by third-party applications. Another example includes some application integration patterns where applications exchange data through file deposits.

Watching for file system changes can be implemented easily by periodically looking at the entries of a folder and determining which files have been added, removed, or changed based on their last modification time stamp. However,

LISTING 19 / LISTING 20 / LISTING 21 / LISTING 22 / LISTING 23 / LISTING 24

```
Posix permissions for asm-all-2.2.3.jar
-> OTHERS_READ
-> OWNER_READ
-> GROUP_READ
-> OWNER_WRITE
```

Download all listings in this issue as text

more-efficient mechanisms exist at the operating system level, such as inotify for Linux.

NIO.2 provides a useful API that takes advantage of operating system facilities, when available, and falls back to periodical polling otherwise. WatchService provides a service that can be notified about changes for a given Path object. You can specify which types of events should be looked for, namely additions, modifications, removals, or overflow (failure of the notification system to report all events). For example, the snippet in **Listing 25** looks for changes in /tmp and prints out events.

A WatchService is obtained from the file system of the path to be monitored. Again, this is an AutoCloseable object, and it is a good idea to use it with a try-with-resources statement to ensure that it eventually gets closed. The watch service is then registered to a path, and events become available from a queue.

A WatchKey is used in conjunction with registration and is returned at registration time. A WatchKey is also obtained when looking for events by using either the watch service's poll() or take() method. In the example in **Listing 25**, we opted to use take() because we can afford to wait for a new event to be available. A watch key maintains state, so we can query to determine whether it is still valid and to obtain the latest watch events.

FILTER AND MATCH

Powerful mechanisms are provided with the java.nio.file.Files class: file visitors and directory streams, which can perform selective filtering and matching for discovered elements.

Once a key has been used, it needs to be reset if it is to be used again. The reset() method returns a Boolean value that will be false if the key is no longer valid, for instance, when a watched folder disappears. You can also call the cancel() method to cancel the watch service registration.

Finally, the key has information about the event. For example, it could hold information on the kind of event (most likely from StandardWatchEventKinds) and a context object that is an instance of Path, which corresponds to the kind of events the service was looking for (ENTRY_CREATE, ENTRY_MODIFY, and ENTRY_DELETE).

LISTING 25

```
import static java.nio.file.StandardWatchEventKinds.*;

// (...)

Path tmp = Paths.get("/tmp");
try (WatchService watcher = tmp.getFileSystem().newWatchService()) {
 boolean running = true;
 tmp.register(watcher, ENTRY_CREATE, ENTRY_DELETE, ENTRY_MODI-
FY);
 while (running) {
  WatchKey key = watcher.take();
  for (WatchEvent<?> event : key.pollEvents()) {
   Path path = (Path) event.context();
   WatchEvent.Kind<?> kind = event.kind();
   System.out.println(kind + " => " + path);
  }
  running = key.reset();
 }
}
```

Download all listings in this issue as text

Conclusion

This article introduced the new file system APIs released in Java SE 7 as part of the so-called NIO.2 packages. These packages make it easy to deal with both common and advanced file operations, such as efficiently monitoring changes in a folder or moving files into a compressed zip archive while preserving metadata. Last but not least, they integrate well with existing java.io and java.nio classes, and they can be extended to uniformly deal with new types of file system providers. For more examples, see the sample NIO.2 code in the JDK 7 distribution (under sample/nio/file). </article>

LEARN MORE

- Java SE 7 API
- "File I/O (Featuring NIO.2)" Java Tutorial
- NIO Project for OpenJDK
- "Better Resource Management with Java SE 7: Beyond Syntactic Sugar"

ADAM BIEN

BIO

PHOTOGRAPH BY THOMAS EINBERGER/ GETTY IMAGES

Java EE Connector Architecture 1.6

Java EE Connector Architecture 1.6 provides deep integration with container services in a lean way.

Mainstream resources, such as persistence or messaging, can be accessed directly with dependency injection (DI) or JNDI lookup. You can easily integrate nonstandard resources so that they can also be injected in the application, but container services such as pooling, monitoring, security, or fine-grained transactions might be unavailable.

Implementing container services for each resource type is a complex task. On the other hand, the Java EE Connector Architecture (JCA) provides a standard mechanism through which these container services can be used more simply.

This article covers the implementation of a file resource adapter that is based on JCA and is in the style of a key-value store, which uses a minimal number of files and no XML, and is configured purely using annotations.

JCA 1.6—Easier to Use than Expected

The aim of JCA 1.6 is as follows:

"*The Java EE Connector architecture defines a standard architecture for connecting the Java EE platform to heterogeneous EISs. Examples of EISs include Enterprise Resource Planning (ERP), mainframe transaction processing (TP), and database systems. The connector architecture defines a set of scalable, secure, and transactional mechanisms that enable the integration of EISs with application servers and enterprise applications.*" (JCA 1.6 specification, JSR 322)

Although JCA was originally meant to integrate heavy enterprise systems, it is simple enough to use for accessing any external systems. A handful of classes are sufficient to implement a fully functional resource adapter (connector). Version 1.6 of the JCA specification makes using XML deployment descriptors for configurations optional by introducing annotation equivalents, which makes the developer's task easier. Direct access to container services—such as work management, security, or transaction state—makes using the connector implementation easier than implementing those services within an application.

MAKE IT SIMPLE
JCA 1.6 provides a **standard mechanism** through which container services can be used more simply.

In this article, we will implement an outbound connector that accesses a local file system with the API shown in **Listing 1**.

The API interface defined in **Listing 1** is going to be directly injected into the application code. As with JDBC, the JCA outbound artifacts are connection-oriented and, hence, connections need to be obtained and returned to the pool (for example, by calling connection.close()). A Bucket instance can be created by injecting BucketStore in the component bean (EJB bean/Servlet), as shown in **Listing 2**.

Both files reside in dedicated JAR package-based Maven projects and are ready to be referenced by applications (see **Listing 3**).

To compile the API, the interface javax.resource.Referenceable is needed, which is part of the JCA API. Referenceable comes with the Java EE 6 API as a GlassFish 3.1.2 embedded library, which is referenced once in the file's parent project. At the same time, the parent is used to build all modules consistently, as shown in **Listing 4**.

Most connector implementations are divided into the API and implementation modules. The Bucket interface belongs to the API and is built with the jar packaging type. The API module is built by a project object model (POM) with the jar packaging type, but it is enhanced with the resource adapter archive (RAR) plug-in, as shown in **Listing 5a** and **Listing 5b**.

Both the application and the resource adapter implementation rely on the API classes for compilation. The parent project declares only the common Java EE 6 API dependency and simplifies the consistent building of both resource adapter modules. The client module is an optional submodule and can be used for testing and demo purposes.

The Necessary Plumbing

A resource adapter (connector) implementation can be bidirectional between the external system and the application server. The application server notifies the resource adapter about the state of transactions, connection lifecycle, and security.

On the other hand, the application server expects the resource adapter to expose a minimal set of metadata. Starting with JCA 1.6, the metadata can be provided as annotations instead of XML deployment descriptors (ra.xml). The FileAdapter class has the component-defining annotation (@Connector), which primarily indicates that the bundle is a resource adapter (apart from being packaged in .rar format). See **Listing 6a** and **Listing 6b**.

We don't need a reference to the BootstrapContext and we are not interested in other callback invocations in this case, so all the methods remain empty.

Starting with the Meat . . .

The implementation of BucketStore is trivial, as shown in **Listing 7a** and **Listing 7b**.

The FileBucketStore acts as a ConnectionFactory, so it also has to implement the Referenceable interface from the BucketStore. Both the Referenceable and Serializable interfaces ensure proper registration in the JNDI space, as stated in the API.

The implementation of the Referenceable interface only requires it to implement an accessor method pair to maintain the javax.naming.Reference instance. More interesting is the factory method getBucket. Instead of creating a new bucket for each call, we are asking the ConnectionManager to allocate a connection with the passed ManagedConnectionFactory.

An anonymous instance of the ConnectionRequestInfo is passed as a second parameter to the allocateConnection method. Hence, all connections are considered as equal, and both the equals and hashCode methods are implemented to attest to the equality of all ConnectionRequestInfo instances. Future versions of the connector, however, could extend the implementation to support authentication. You can also pass "null" instead of the anonymous ConnectionRequestInfo, but this might result in unportable code.

The entire transactional interaction with the local file system is implemented in the FileBucket class. In addition to the Bucket interface from the connector's API, FileBucket also imple-

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

```
public interface Bucket extends AutoCloseable{

    void write(String file,byte[] content);
    void delete(String file);
    byte[] fetch(String file);
    @Override
    void close();

}
```

Download all listings in this issue as text

ments the close method from the AutoCloseable interface. With the implemented AutoCloseable interface, the application can rely on the try-with-resources mechanism to close connections that are not needed. See **Listing 8a–Listing 8e**.

However, the FileBucket class delegates only all close() invocations immediately to the Closeable interface instance passed in the constructor. With the begin, commit, and rollback callbacks, the FileBucket is informed about the transaction progress.

Applications will use the FileBucket implementation to access the content in the local file system. Write and delete operations are not performed directly; they are cached. During transaction commit, the method flushChanges is invoked to persist the cache to the local file system.

A rollback invocation, however, clears the cache and does not affect the persistence store (local file system). As the transaction begins, the method createIfNotExist is invoked to create the root folder in the working directory.

The Reusable Parts

FileBucketStore is a factory of the FileBucket class. Both classes realize the file access functionality and are, therefore, highly domain-specific implementations. FileBucket implements the file access without any dependency on the JCA API and can be easily unit tested, as shown in **Listing 9a** and **Listing 9b**.

Both remaining classes are required implementations of the ManagedConnection and ManagedConnectionFactory interfaces from the javax.resource.spi package. GenericManagedConnection implements the ManagedConnection interface and represents the physical connection effectively managing a FileBucket instance. As the name implies, the GenericManagedConnection is only weakly dependent on any domain-specific implementation and can be reused for different resource adapter (connector) implementations. See **Listing 10a–Listing 10f**.

Most of the GenericManagedConnection methods are self-explanatory. The essential functionality is implemented in the fireConnectionEvent method. Events fired in this method notify the application server about the state of the ManagedConnection. With this callback mechanism, the application server is able to properly manage and clean up the connection pool. Without the event notification mechanism, the managed connection pool would run dry and then block the execution of the application. Because the sample connector supports only local transactions, the method getXAResource has to throw a ResourceException to indicate the lack of Extended Architecture (XA)–transaction support.

A bit more challenging is the implementation of the

LISTING 6a / LISTING 6b / LISTING 7a / LISTING 7b

```
import javax.resource.ResourceException;
import javax.resource.spi.ActivationSpec;
import javax.resource.spi.BootstrapContext;
import javax.resource.spi.Connector;
import javax.resource.spi.ResourceAdapter;
import javax.resource.spi.ResourceAdapterInternalException;
import javax.resource.spi.TransactionSupport;
import javax.resource.spi.endpoint.MessageEndpointFactory;
import javax.transaction.xa.XAResource;

@Connector(reauthenticationSupport = false,
transactionSupport =
TransactionSupport.TransactionSupportLevel.LocalTransaction)
```

Download all listings in this issue as text

GenericManagedConnectionFactory:, which is shown in **Listing 11a**, **Listing 11b**, and **Listing 11c**.

Beyond trivial bookkeeping, the GenericManagedConnection class provides valuable metadata, such as an @ConnectionDefinition annotation that represents an outbound artifact. The application server uses the information provided in the annotation to find the implementation of the domain-specific connector. Also interesting are the annotations on the setRoot Directory method. The @ConfigProperty annotation indicates that the property is configurable by a resource adapter user and is used to generate a convenient user interface. The @Min annotation from the Bean Validation specification (JSR 303) is used to validate the user input before setting the values.

In the method match ManagedConnection, a GenericManagedConnection instance has to be identified and extracted from the set of all known connections. The ConnectionRequestInfo parameter is intended to find its corresponding GenericManagedConnection instance.

A Key-Value Store with Resource Adapter

FilesResource exposes the connector's functionality as a RESTful service implemented as an EJB bean. The BucketStore can be conveniently injected with the @Resource annotation into any Java EE component, as shown in **Listing 12a** and **Listing 12b**.

The injected BucketStore instance acts as a Bucket factory. In the FilesResource example, the Bucket instance is declared in a try-with-resources statement. Because the Bucket instance implements java.lang.AutoCloseable, it will be closed regardless of whether the try statement completes normally or abruptly.

BucketStore can be injected with an @Resource annotation containing a valid JNDI name of the connector resource. The JNDI name needed for injection is configured during the connector's deployment.

Deploying and Installing the Connector

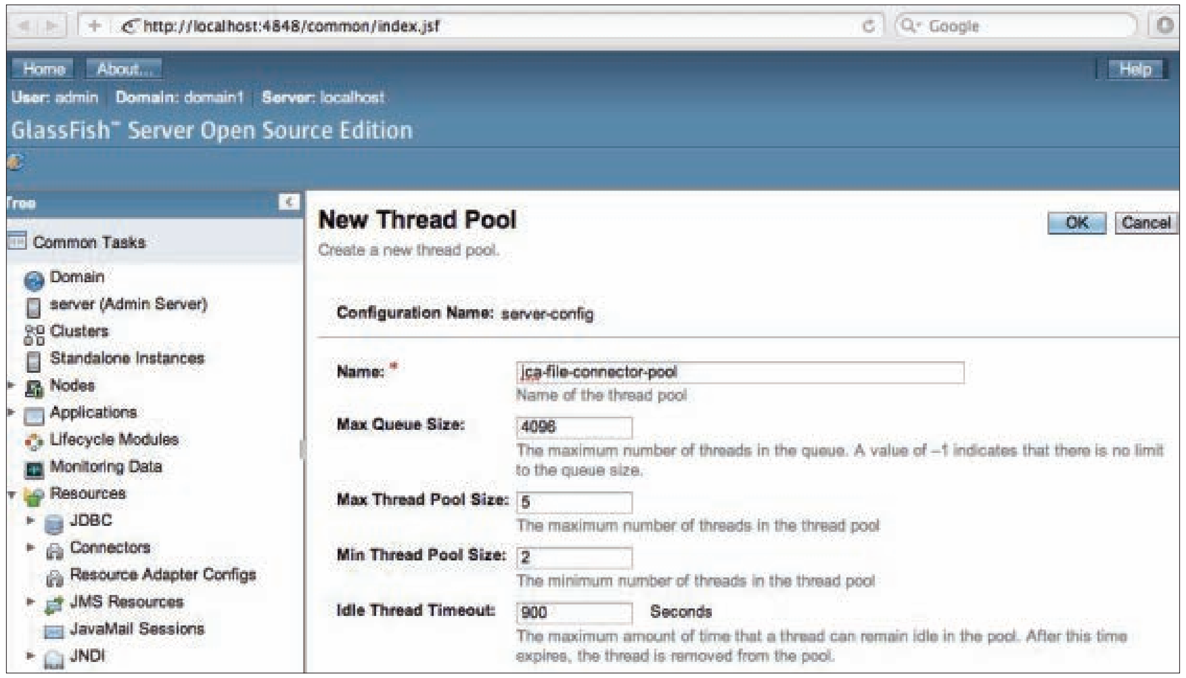
A standalone connector will be packaged as a RAR file and needs to be deployed first. In GlassFish, you can deploy the RAR using the following command-line interface (CLI) command or through the Web-based administration console.

```
$GLASSFISH_HOME/bin/asadmin
--port 4848 deploy --force ./
store/target/jca-file-store.rar
```

LISTING 8a / LISTING 8b / LISTING 8c / LISTING 8d / LISTING 8e

```
import java.io.*;
import java.nio.file.Files;
import java.nio.file.LinkOption;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.Map.Entry;
import java.util.Set;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.ConcurrentSkipListSet;
import javax.resource.ResourceException;
import org.connectorz.files.Bucket;

public class FileBucket implements Bucket {

  private String rootDirectory;
  private ConcurrentHashMap<String, byte[]> txCache;
  private Set<String> deletedFiles;
  private Closeable closeable;
  private PrintWriter out;

  public FileBucket(PrintWriter out, String rootDirectory,
Closeable closeable) {
    this.out = out;
    this.rootDirectory = rootDirectory;
    this.closeable = closeable;
    this.txCache = new ConcurrentHashMap<>();
    this.deletedFiles = new ConcurrentSkipListSet<>();
  }
```

Download all listings in this issue as text



New Thread Pool

Create a new thread pool.

Configuration Name: server-config

Name: * ica-file-connector-pool
Name of the thread pool

Max Queue Size: 4096
The maximum number of threads in the queue. A value of –1 indicates that there is no limit to the queue size.

Max Thread Pool Size: 5
The maximum number of threads in the thread pool

Min Thread Pool Size: 2
The minimum number of threads in the thread pool

Idle Thread Timeout: 900 Seconds
The maximum amount of time that a thread can remain idle in the pool. After this time expires, the thread is removed from the pool.

Figure 1

After you successfully deploy the connector, it is still not usable until you complete the configuration that is required for integration with the application server and the clients. This task might involve setting appropriate external endpoint information, such as the host name, port, user, password, and so on.

Step 1: Configure a thread pool. A dedicated thread pool significantly simplifies management and monitoring. The New Thread Pool wizard in the GlassFish GUI (shown in **Figure 1**) allows you to create and configure a new thread pool.

The file connector does not start any new threads, and it does not rely on the WorkManager functionality. The file system is accessed without involving the WorkManager, so in this case, the WorkManager settings are not required. However, an asynchronous file connector implementation would usually rely on a dedicated thread pool.

Step 2: Configure a resource adapter. After you create the pool using the default values, you need to use the New Resource Adapter Config wizard (shown in **Figure 2**) to create a new resource adapter configuration and point to the thread pool that was created. This instantiates the connector, but the connector is still not visible to applications. However, the connector becomes available for further configuration after this step.

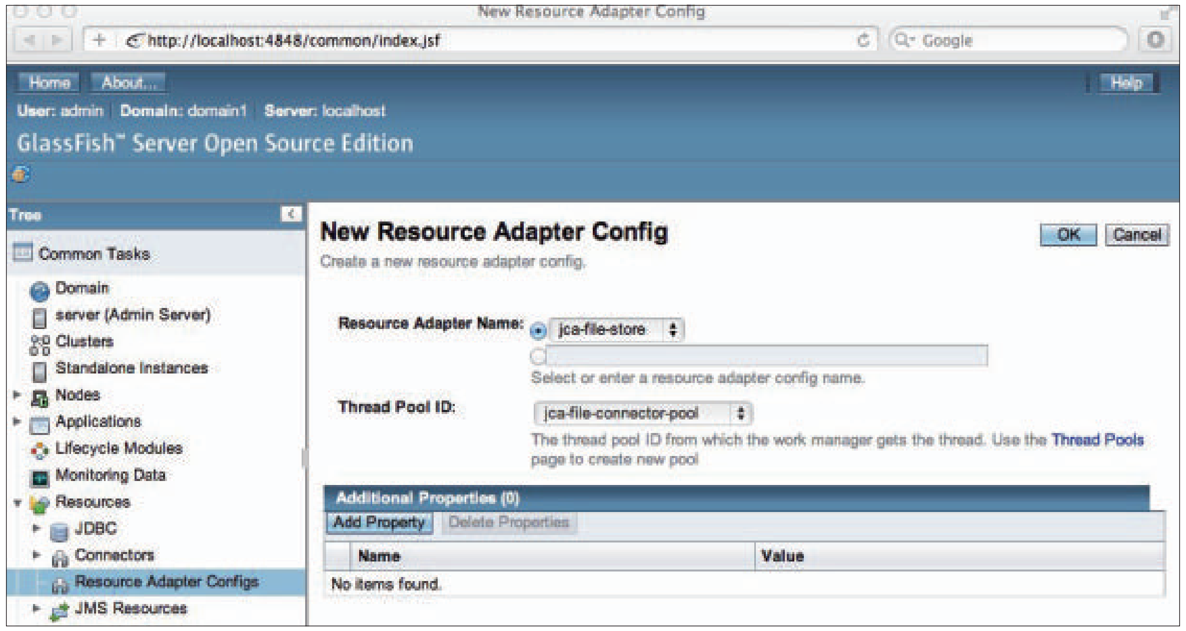
LISTING 9a LISTING 9b

```
public class FileBucketTest {

   FileBucket cut;
   String directory = "./current";
   Closeable closeable;

   @Before
   public void initialize() {
      this.closeable = mock(Closeable.class);
      this.cut = new FileBucket(new PrintWriter(System.out),
directory, this.closeable);
   }

   @Test
   public void autoClose() throws Exception {
      try (FileBucket bucket = new FileBucket(
new PrintWriter(System.out), directory, this.closeable);) {
         bucket.begin();
      }
      verify(this.closeable).close();
   }

   @Test
   public void writeAndRollback() throws Exception{
      final String key = "hey";
      this.cut.begin();
      final byte[] content = "duke".getBytes();
      this.cut.write(key, content);
      byte[] actual = this.cut.fetch(key);
      assertThat(actual,is(content));
      this.cut.rollback();
      actual = this.cut.fetch(key);
      assertNull(actual);

   }
```

Download all listings in this issue as text

Step 3: Create a connection pool. After you configure the resource adapter, you need to create a connection pool for the BucketStore factory, as shown in **Figure 3**.

In this step, from the previously declared connector named jca-file-store, you choose the BucketStore factory, which represents a connection factory. Note that BucketStore is declared as a connection factory in the



New Resource Adapter Config

Create a new resource adapter config.

OK Cancel

Resource Adapter Name: jca-file-store

Select or enter a resource adapter config name.

Thread Pool ID: jca-file-connector-pool

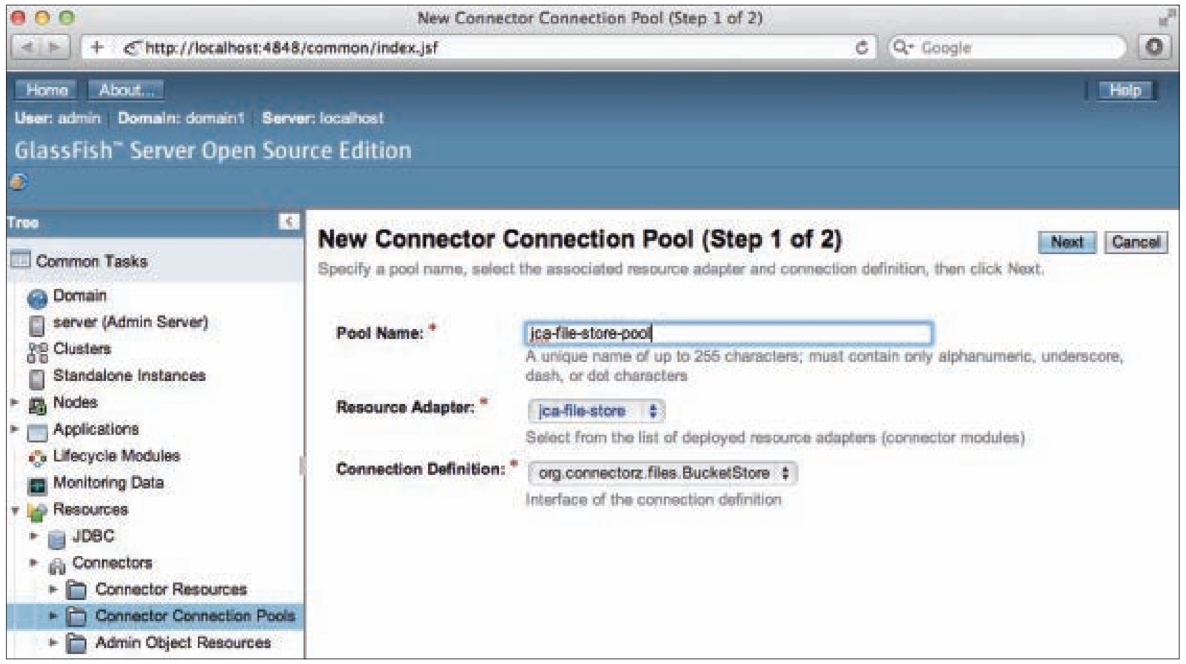
The thread pool ID from which the work manager gets the thread. Use the Thread Pools page to create new pool

Additional Properties (0)

Add Property   Delete Properties

Name	Value
No items found.	

Figure 2



New Connector Connection Pool (Step 1 of 2)

Specify a pool name, select the associated resource adapter and connection definition, then click Next.

Next Cancel

Pool Name: * jca-file-store-pool

A unique name of up to 255 characters; must contain only alphanumeric, underscore, dash, or dot characters

Resource Adapter: * jca-file-store

Select from the list of deployed resource adapters (connector modules)

Connection Definition: * org.connectorz.files.BucketStore

Interface of the connection definition

Figure 3

LISTING 10a / LISTING 10b / LISTING 10c / LISTING 10d / LISTING 10e / LISTING 10f

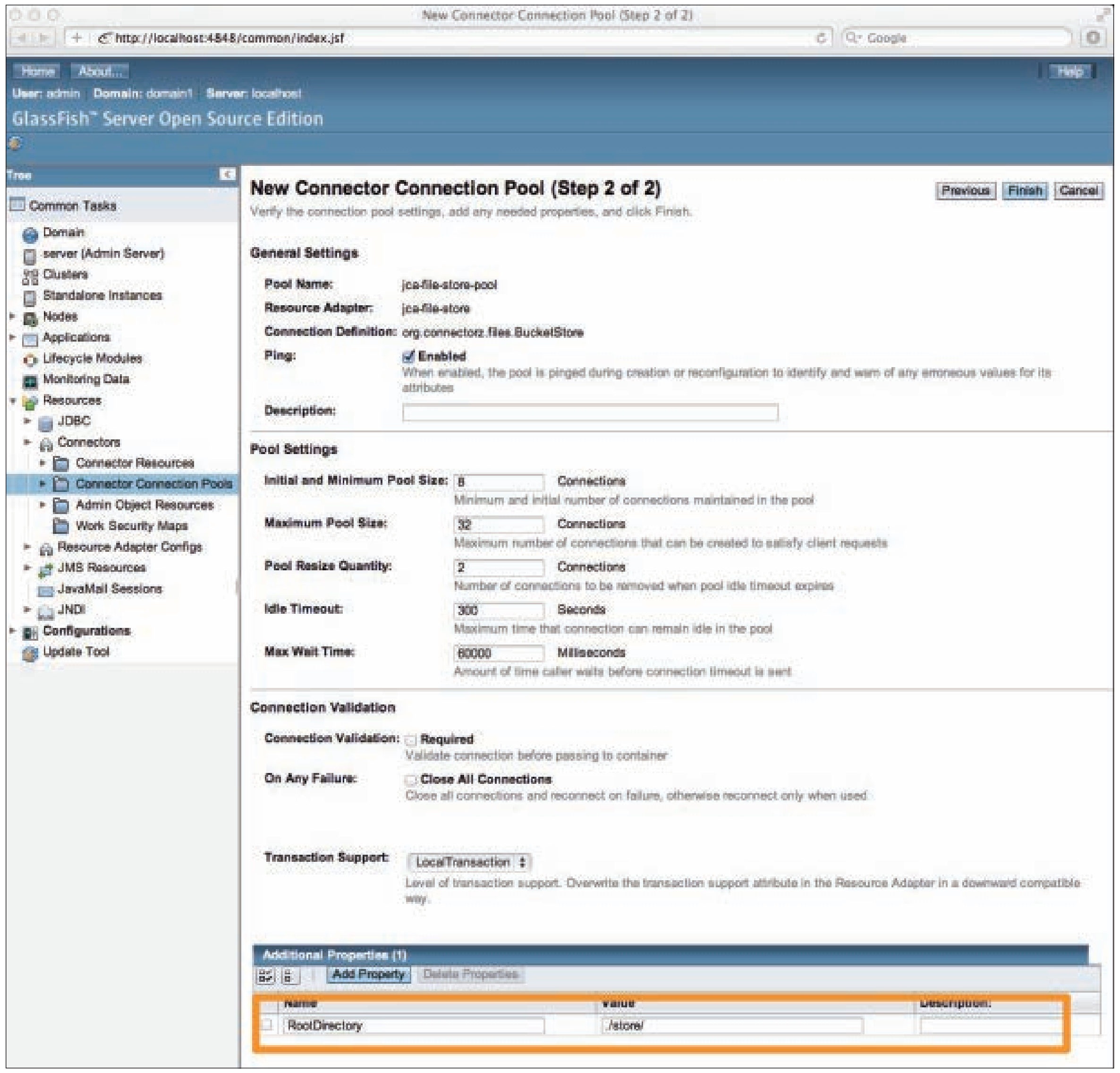
```
import java.io.Closeable;
import java.io.PrintWriter;
import java.util.LinkedList;
import java.util.List;
import javax.resource.ResourceException;
import static javax.resource.spi.ConnectionEvent.*;
import javax.resource.spi.*;
import javax.security.auth.Subject;
import javax.transaction.xa.XAResource;
public class GenericManagedConnection
    implements ManagedConnection, LocalTransaction,
Closeable {

  private ManagedConnectionFactory mcf;
  private PrintWriter out;
  private FileBucket fileConnection;
  private ConnectionRequestInfo connectionRequestInfo;
  private List<ConnectionEventListener> listeners;
  private final String rootDirectory;

  GenericManagedConnection(PrintWriter out,String rootDirectory,Man
agedConnectionFactory mcf,
ConnectionRequestInfo connectionRequestInfo) {
    this.out = out;
    this.rootDirectory = rootDirectory;
    this.mcf = mcf;
    this.connectionRequestInfo = connectionRequestInfo;
    this.listeners = new LinkedList<>();
    this.fileConnection =
new FileBucket(out,this.rootDirectory,this);
  }
```

Download all listings in this issue as text

GenericManagedConnectionFactory instance, and the application server will detect it appropriately. For reference, the @ConnectionDefinition annotation defined in the GenericManagedConnectionFactory is shown in **Listing 13**.

Right after declaring the connector connection pool, you will have to specify the pool settings, as shown in **Figure 4**.

It is difficult to predict optimal values, so it is far easier to start with the defaults and continuously tune the settings after performing stress tests. It is interesting to note that GlassFish also recognizes custom configuration entries declared in the GenericManagedConnectionFactory (see **Listing 14**).


New Connector Connection Pool (Step 2 of 2)
http://localhost:4848/common/index.jsf

User: admin   Domain: domain1   Server: localhost
GlassFish™ Server Open Source Edition

New Connector Connection Pool (Step 2 of 2)
Verify the connection pool settings, add any needed properties, and click Finish.

General Settings

Pool Name:	jca-file-store-pool
Resource Adapter:	jca-file-store
Connection Definition:	org.connectorz.files.BucketStore
Ping:	✓ Enabled — When enabled, the pool is pinged during creation or reconfiguration to identify and warn of any erroneous values for its attributes
Description:	

Pool Settings

Initial and Minimum Pool Size:	8	Connections — Minimum and initial number of connections maintained in the pool
Maximum Pool Size:	32	Connections — Maximum number of connections that can be created to satisfy client requests
Pool Resize Quantity:	2	Connections — Number of connections to be removed when pool idle timeout expires
Idle Timeout:	300	Seconds — Maximum time that connection can remain idle in the pool
Max Wait Time:	60000	Milliseconds — Amount of time caller waits before connection timeout is sent

Connection Validation

Connection Validation:	☐ Required — Validate connection before passing to container
On Any Failure:	☐ Close All Connections — Close all connections and reconnect on failure, otherwise reconnect only when used
Transaction Support:	LocalTransaction — Level of transaction support. Overwrite the transaction support attribute in the Resource Adapter in a downward compatible way.

Additional Properties (1)

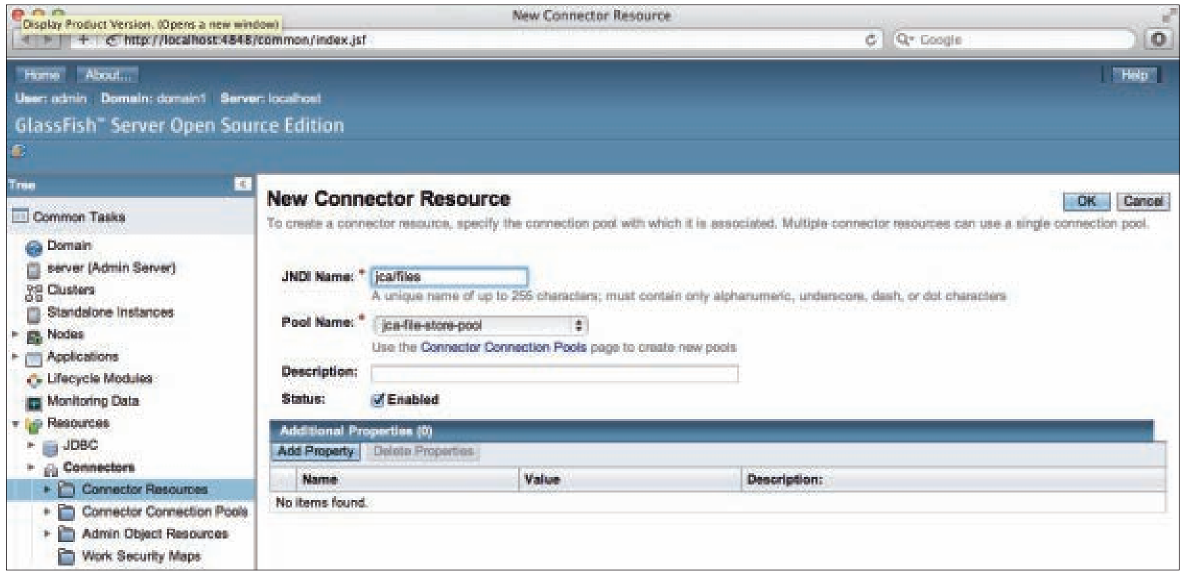
Name	Value	Description:
RootDirectory	./store/	

Figure 4

LISTING 11a / LISTING 11b / LISTING 11c / LISTING 12a / LISTING 12b

```
import org.connectorz.files.BucketStore;
import java.io.PrintWriter;
import java.io.Serializable;
import java.util.Iterator;
import java.util.Objects;
import java.util.Set;
import javax.resource.ResourceException;
import javax.resource.spi.*;
import javax.security.auth.Subject;
import javax.validation.constraints.Min;
import org.connectorz.files.Bucket;

@ConnectionDefinition(connectionFactory = BucketStore.class,
  connectionFactoryImpl = FileBucketStore.class,
  connection = Bucket.class,
  connectionImpl = FileBucket.class)
public class GenericManagedConnectionFactory
    implements ManagedConnectionFactory, Serializable {

  private PrintWriter out;
  private String rootDirectory;

  public GenericManagedConnectionFactory() {
    out = new PrintWriter(System.out);
  }

  @Min(1)
  @ConfigProperty(defaultValue = "./store/",
supportsDynamicUpdates = true, description = "The root folder
of the file store")
  public void setRootDirectory(String rootDirectory) {
    this.rootDirectory = rootDirectory;
  }
```

Download all listings in this issue as text


New Connector Resource

To create a connector resource, specify the connection pool with which it is associated. Multiple connector resources can use a single connection pool.

JNDI Name: * jca/files
A unique name of up to 255 characters; must contain only alphanumeric, underscore, dash, or dot characters

Pool Name: * jca-file-store-pool
Use the Connector Connection Pools page to create new pools

Description:

Status: ✓ Enabled

Additional Properties (0)

	Name	Value	Description:
	No items found.		

Figure 5

With the exposed rootDirectory property in the GenericManagedConnectionFactory, you can set the location of the root folder directly from the Web-based administration console. The administration view is generated from the setRootDirectory method's metadata, for example, the name, the parameter type, and the @ConfigProperty and @Min annotations. Interestingly, the Bean Validation annotation (JSR 303) can be used to verify user input and to prevent invalid entries.

Step 4: Specify the JNDI name. The last step is to register the freshly configured connection pool with the custom JNDI name, as shown in **Figure 5**. Java EE components can use the specified JNDI name for dependency injection of the BucketStore instance.

Conclusion

After the configuration of all custom values, the file connector's resources can be injected into Java EE components with the preconfigured JNDI name. With the exposure as a RESTful resource, all the JCA file connector functionality becomes available to any HTTP clients. For Java clients, the easiest way to interact with the FilesResource is to use the standardized JAX-RS 2.0 client (see **Listing 15a** and **Listing 15b**.)

A JCA connector implementation, comprising a few classes and no XML deployment descriptors, allows you to "legally" access files in a Java EE environment. But the actual benefit is having access to the files in a transactional, secure, managed, and monitorable way.

In addition, you can easily throttle the concurrency to the exposed resource by configuring the resource adapter's connection pool. Resource adapter monitoring information (connection pool, work manager, and so on) is also exposed by the application server. GlassFish makes the monitoring data available through RESTful services, which simplifies management and monitoring. `</article>`

LISTING 13 / LISTING 14 / LISTING 15a / LISTING 15b

```
@ConnectionDefinition(connectionFactory = BucketStore.class,
  connectionFactoryImpl = FileBucketStore.class,
  connection = Bucket.class,
  connectionImpl = FileBucket.class)
public class GenericManagedConnectionFactory{}
```

Download all listings in this issue as text

LEARN MORE

- connectorZ
- JSR 322: Java EE Connector Architecture 1.6
- JSR 303: Bean Validation
- "Generic JCA" chapter of *Real World Java EE Patterns—Rethinking Best Practices* (press.adam-bien.com, 2011)

VIKRAM GOYAL

BIO

Payment API—Getting Started with JSR 229

Learn to use JSR 229 in Java ME applications and games.

In-app transactions allow gamers and app users to enjoy a seamless payment transaction without needing to leave the game or application. In this article, I will cover the Payment API (JSR 229) available to developers of Java ME applications and games and show how to use it with the help of a sample application that simulates the Payment API in action.

Note: The source code for the sample application described in this article can be downloaded as a NetBeans project here.

Payment API Background

The Payment API defined by JSR 229 has been kept very simple to allow for various interpretations. This is a good thing, because the API makes it easy to create *adapters*, which are what drive the underlying payment processing from within a game or application.

The API is defined in the javax.microedition.payment package and contains two interfaces, one concrete class, and four exception classes.

TransactionRecord and TransactionListener Interfaces

The interface TransactionRecord represents a record of a transaction undertaken by the Payment API's modules. To represent the state of these records, it uses an integer flag with one of these values: TRANSACTION_SUCCESSFUL, TRANSACTION_FAILED, or TRANSACTION_REJECTED. In addition, it contains five informational methods that provide metadata about the record:

- int getFeatureID()—A feature is something that can be charged for or must be paid for (for example, a higher level in a game or a value-added service). Each feature must be assigned a unique identifier, and this method returns the value of the identifier (as provided by the application developer to the payment module). So, each transaction record must be associated with an application feature.
- int getTransactionID()—This is a platform-unique identifier for the transaction record.
- int getState()—This indicates whether the transaction was completed successfully, failed, or was rejected by the user.
- long getFinishedTimestamp()—This indicates at what time the payment module acknowledged this transaction.
- boolean wasMissed()—This indicates whether the transaction record was processed in a previous session rather than in the current session.

The interface TransactionListener is a simple listener interface that enables developers to know when a transaction has been processed (successfully or otherwise). The transaction module does the processing asynchronously and pings the listener when

KEEP IT SIMPLE
The Payment API has been **kept very simple** to allow for various interpretations.

PHOTOGRAPH BY JONATHAN WOOD/ GETTY IMAGES it is done. This interface defines only one method: processed(TransactionRecord record).

TransactionModule Class

The TransactionModule class calls the processed(TransactionRecord record) method of the TransactionListener when the transaction has been performed (successfully or otherwise). The record holds all the data and the state of the transaction.

In a nutshell, the TransactionModule class is the go-between for your application and the underlying payment module. It defines the process(int featureID, String featureTitle, String featureDescription) and process(int featureID, String featureTitle, String featureDescription, byte[] payload) methods that your application calls each time it wants to charge the user for a new feature that the user must pay for. These methods return immediately, and the state of the transaction is sent to any listener registered by the module, as discussed in the previous section.

ABOUT THE API

The API simply leaves the actual payment processing part up to the underlying adapters.

Calling either of these two methods causes the payment module to interact with the end users by creating a GUI that shows the featureTitle and featureDescription, along with the price of the requested feature so users can confirm that they would like to make the purchase. This visual request is accompanied by a few other fields that validate the accuracy of the payment.

Of course, this display occurs only if the data that is passed to either of the process methods is valid. For example, if either featureTitle or featureDescription is null or empty, the method will instead throw a TransactionModuleException. If a listener hasn't been registered by calling the setListener() method before either of the process methods is called, the method will throw a TransactionListenerException. As we will see, a TransactionFeatureException is thrown if the payment module doesn't know about this feature via the Java archive (JAR) manifest file.

The second process method accepts a payload in the form of a byte[] array, which can be sent on to the payment processing adapters for extra information about the transaction. A TransactionPayloadException is thrown if the underlying adapter cannot process this payload.

Finally, these methods return a unique int identifier that can be used to identify and differentiate this payment request from others.

The TransactionModule class defines only one constructor, TransactionModule(Object obj), which normally takes a reference to the MIDlet it is running in. If there is a problem with the underlying payment module not being correctly set up, calling this constructor will throw a TransactionModuleException. This constructor is the starting point for all payment-related operations.

The TransactionModule class also defines the deliverMissedTransactions() method, which is used in case an application crash occurs after a process() call was initiated but before the listener's processed() method could be called. This method then attempts to deliver all these missed transactions.

One last method is the getPastTransactions(int max) method, which returns an array of TransactionRecords that have been processed by the current application.

Provisioning Adapters and Features

All this information about the API is important, but it doesn't fully answer the question about how and when the application processes the payment. The API simply leaves the actual payment processing part up to the underlying adapters. Device manufacturers provide these adapters, regardless of whether they are a premium-priced SMS adapter or an adapter to charge a payment to a credit card.

As an application developer, you have little control over adapters. You can request that a particular payment adapter be present, but you cannot mandate it. If none of your requested adapters are available or configured correctly, your application will not run on the device. So well in advance, you need to be sure that the target devices for your application or game will support the payment adapters.

However, you have full control over the provisioning of your features, which is done via the JAR manifest file of your MIDlet application. In this section, I will show how to define features and how they translate to payments in the underlying Java Application Descriptor (JAD) and JAR manifest files.

Let's assume that you are creating an application that is based on the *freemium* model, in which the basic app and some features are

PLAN	PRICING
FREE	FREE
BUSINESS	$5.00
PROFESSIONAL	$10.00

Table 1

free but premium features require payment. Let's further assume that there are three pricing plans, as shown in **Table 1**.

Defining Adapters

To get started, you need to define at least two entries in your JAD file: Pay-Version and Pay-Adapters.

Pay-Version is simply the version that correlates the JAD file and the JAR manifest file. It is usually 1.0.

Pay-Adapters is a list of comma-separated adapters that your application is happy to work with. At least one of these adapters must be supported on the target device or your application cannot run. This information is defined in the JAD file because the JAD file is what is read by a device before installing a MIDlet and the device can, therefore, reject the MIDlet if none of the adapters are supported on the device.

An example JAD should, minimally, contain entries like this:

```
Pay-Version: 1.0
Pay-Adapters: PPSMS, X-CCARD
```

PPSMS and X-CCARD are names of payment adapters that are supported by the Java ME toolkits. PPSMS refers to a premium-priced SMS adapter, while X-CCARD is a nonstandard credit card adapter. The Payment API mandates that all devices that implement the Payment API must, at the very least, support PPSMS.

The specification for the Payment API defines several other entries that can go in the JAD file for debugging and testing.

Defining Payment Providers

Each adapter, as defined in the previous section, can work with multiple providers. For example, Visa, MasterCard, and American Express are examples of payment providers for the payment adapter X-CCARD.

Each provider needs to be defined within the JAR manifest file using the Pay-Providers and the Pay-<ProviderTitle>-Info attributes. Each pay provider listed in the Pay-Providers attribute must have a corresponding Pay-<ProviderTitle>-Info attribute. An example JAR manifest file with these entries is shown here:

```
Pay-Providers: ATT, VISA
Pay-ATT-Info: PPSMS, USD, 999, 99
Pay-VISA-Info: X-CCARD, USD, VISA
```

In essence, we are saying that our application works with the payment providers AT&T and Visa, and that AT&T payment is via the premium-priced SMS adapter (defined earlier in the JAD file) and the Visa payment is via credit card (again, as defined earlier in the JAD file).

Further, for each payment provider, we list the currency that is accepted and more payment-specific information. For AT&T, we have listed the Mobile Country Code (MCC) 999 and the Mobile Network Code (MNC) 99.

Connecting Payment Providers to Features

Having defined the payment adapters and the payment providers, it's time to create the correlation between the pricing model that we talked about earlier for our freemium application and the relevant entries in the JAR manifest file. We do this using the Pay-Feature-<n> and the Pay-<ProviderTitle>-Tag-<n> attributes. (Of course, you don't need to define the free pay structure; you need to define only the paid levels.)

Each feature (that is, each payment) maps to each provider title using *tags* that use incremental tag numbers with no gaps and with numbers starting at 0. The first feature is Pay-Feature-0, which must map to each provider title using the tag number 0. See **Table 2** for clarification.

So, for the end user to pay $5.00 for the upgrade to the business level, we define the Pay-Feature-0 attribute with a value of 0. This value is used to map to the two provider titles we defined earlier. Since we don't know which pay-

PLAN	PRICING	Pay-Feature-<n>/VALUE OF THIS ATTRIBUTE	Pay-<ProviderTitle>-Tag-<n>/VALUE OF THIS ATTRIBUTE
FREE	FREE	NOT REQUIRED	NOT REQUIRED
BUSINESS	$5.00	Pay-Feature-0/0	Pay-ATT-Tag-0/5.00, 5550000, 500 Pay-VISA-Tag-0/5.00, 500
PROFESSIONAL	$10.00	Pay-Feature-1/1	Pay-ATT-Tag-1/10.00, 5550000, 1000 Pay-VISA-Tag-1/10.00, 1000

Table 2

ment adapter might be present on the device or which payment type the user might choose, we need to provide this information in the Pay-<ProviderTitle>-Tag-<n> attribute for each provider:

```
Pay-Feature-0: 0
Pay-ATT-Tag-0: 5.00, 5550000, 500
Pay-VISA-Tag-0: 5.00, 500
```

For the AT&T tag, we are saying that $5.00 needs to be charged for the phone number 5550000, and 500 is a reference number. For the Visa tag, we are saying that $5.00 needs to be charged with a reference number 500.

This is the basic information we need to provide in our JAD and JAR manifest files. After this point, it is left to the underlying payment adapter built into the device to inform the user about the payment options (PPSMS or X-CCARD) and to ask for more relevant information (for example, the credit card details).

Updating Payment Information

Things change, and so will your payment options. Due to inflation, you might decide to ask for more next year for the same things. Or you might want to update the list of payment providers that you will accept payments from. The Payment API makes it easy to update your payment information by providing two (mandatory) fields:

```
Pay-Update-Stamp
Pay-Update-URL
```

The first field states the date from which the payment information within the JAR manifest file is valid. The second is a link to a file with the extension .jpp, which defines any updated payment information. This .jpp file is essentially in the same format as the JAR manifest file.

CHANGE IS EASY

Things change, and so will your payment options. You might want to charge more next year. The Payment API makes it easy to update your payment information by providing two (mandatory) fields.

This information is provided to end users each time they make a payment to let them know the last time a payment was generated and to alert them that an update might be available. End users may then choose to check whether an update is available before proceeding with the payment and paying with the updated information.

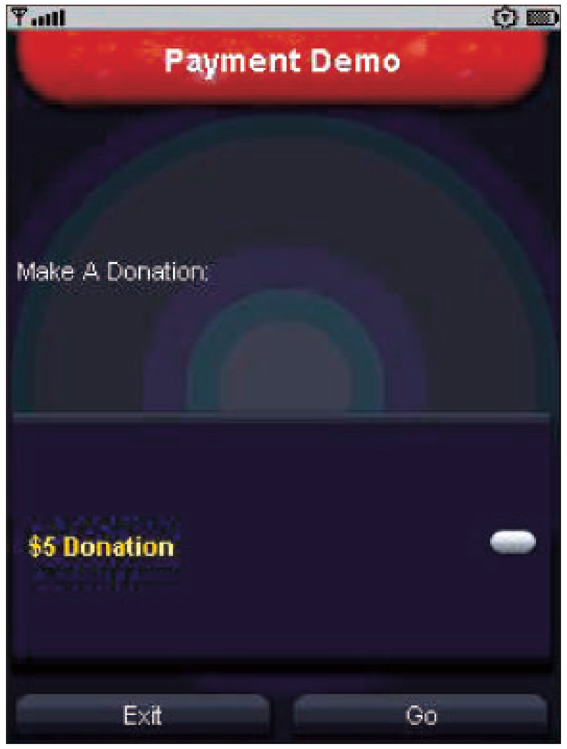
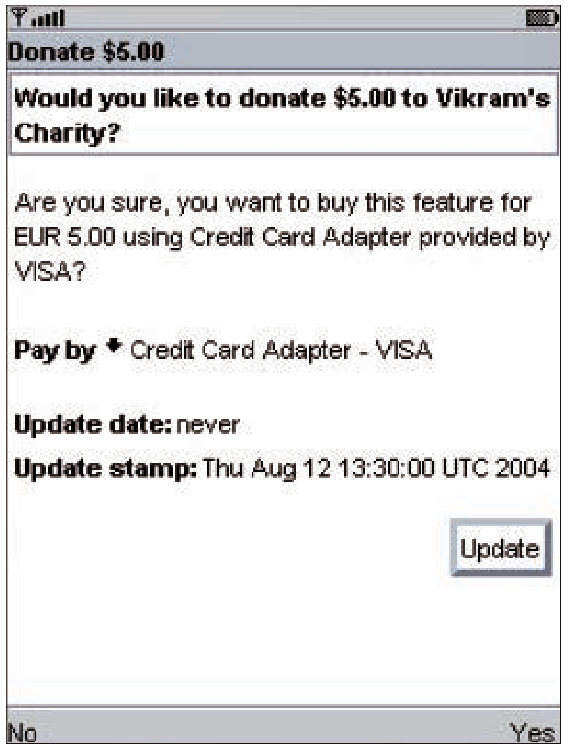


Figure 1



Donate $5.00

Would you like to donate $5.00 to Vikram's Charity?

Are you sure, you want to buy this feature for EUR 5.00 using Credit Card Adapter provided by VISA?

Pay by Credit Card Adapter - VISA

Update date: never

Update stamp: Thu Aug 12 13:30:00 UTC 2004

Figure 2

Sample Application

Before running our sample application, there are some caveats that you should understand.

Different versions of the Oracle emulator provide different payment adapters. The Sun Java Wireless Toolkit for CLDC 2.5.2 from Oracle provides a credit card adapter, which is missing in the Sun Java Wireless Toolkit for CLDC 3.0.5 from Oracle. So, to see the credit card adapter in action, you will need to install the older toolkit and use it as your target platform.

Also, if you use the PPSMS adapter, your target device must be configured with the right MNC and MCC codes. I have used 999 and 99 respectively in my JAR manifest file, and these are the same values that must be present in your target device. How you do this depends on the toolkit that you use. For toolkit version 3.0.5, this information can be seen and updated using the **Java ME -> Device Selector** menu option.

The sample application is fairly simple. **Figure 1** shows a screenshot using the Lightweight User Interface Toolkit (LWUIT) interface.

In the application, I am asking users to donate to, *ahem*, Vikram's

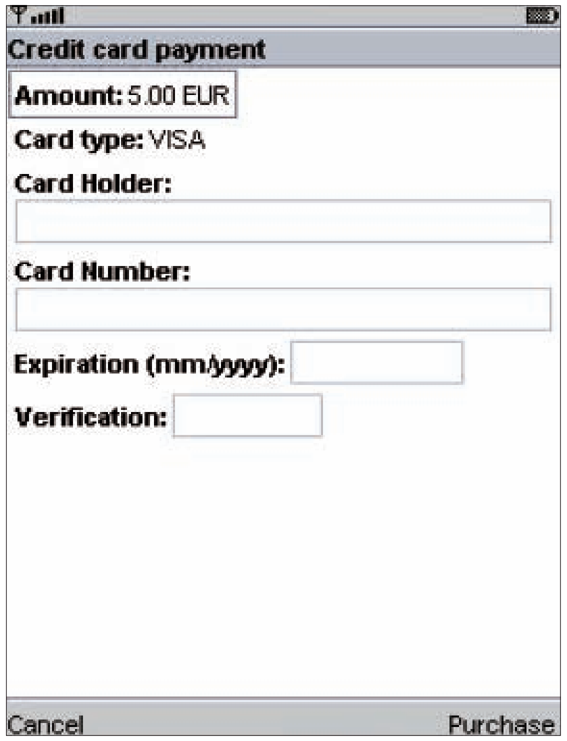
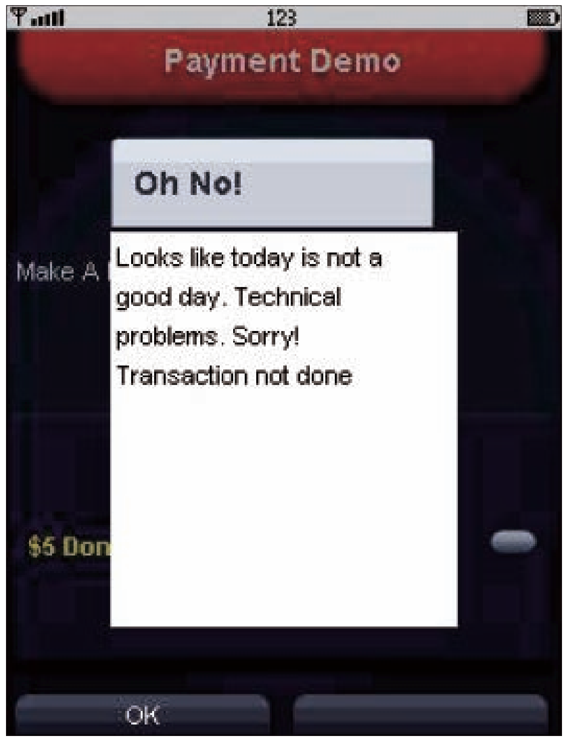

Credit card payment
Amount: 5.00 EUR
Card type: VISA
Card Holder:
Card Number:
Expiration (mm/yyyy):
Verification:

Figure 3



Oh No!
Looks like today is not a good day. Technical problems. Sorry! Transaction not done

Figure 4

LISTING 1 LISTING 2

```
try {
 int selectedId = selection.getSelectedIndex(); // what was selected
 // process it
 module.process(
  selectedId,
  "Donate " + (selectedId == 0 ? "$5.00" : "$10.00"),
  "Would you like to donate " + (selectedId == 0 ? "$5.00" : "$10.00") +
  " to Vikram's Charity?");
 // and then wait.... until we get notified that the
 // payment is processed
 synchronized(this) {
  try {
   wait();
  } catch(InterruptedException ie) {
  }
 }
} catch(Exception ex) {
 handleError(ex);
}
```

Download all listings in this issue as text

Charity. I have provided two payment values or features ($5.00 or $10.00). When the user selects **Go** (after making a value selection), the Payment API takes over because I make the call to the transaction module's process method (see **Listing 1**).

We make the main thread wait until the payment has been processed (approved, rejected, or failed). We will get a call back in the transaction module's processed method (see **Listing 2**).

The first thing we do is wake up the main thread by calling notify(). Finally, we look at the transaction record to see whether the transaction was successful, rejected, or failed and then present a message to the user accordingly.

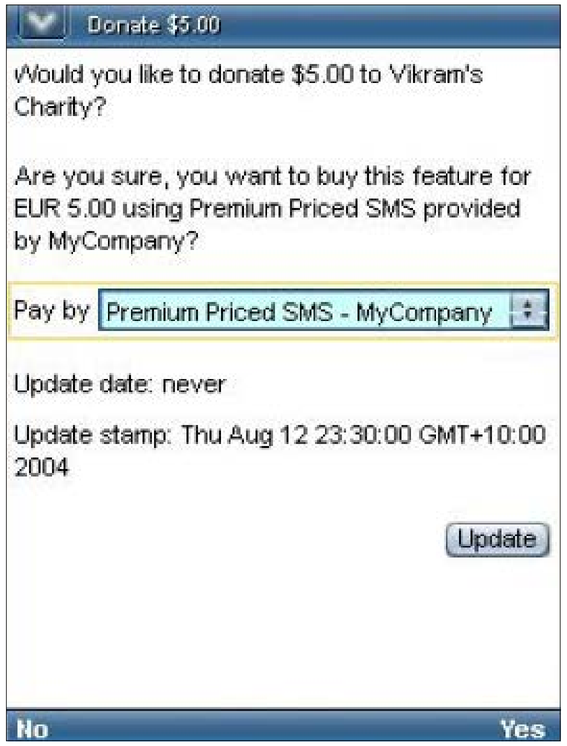
ON THE RISE
The Payment API is **gaining popularity** and is likely to be a popular choice for implementing payment options in the future.

Figure 2 shows the payment information being presented to the user for confirmation of the payment. Note the update data being presented. You have no control over the look and feel of this screen (or the next screen), which is controlled by the underlying adapter. I am using toolkit version 2.5.2 in this example to show the credit card transaction.

Figure 3 shows the credit card information being requested from the user after the user selects **Yes** in **Figure 2**. Again, you have no control over this process or the look and feel of this screen.

Figure 4 shows the result of a transaction that failed due to technical errors.

Finally, **Figure 5** and **Figure 6** show the application being run in toolkit version 3.0.5 using the PPSMS as the underlying adapter, with the transaction being successful.


Donate $5.00

Would you like to donate $5.00 to Vikram's Charity?

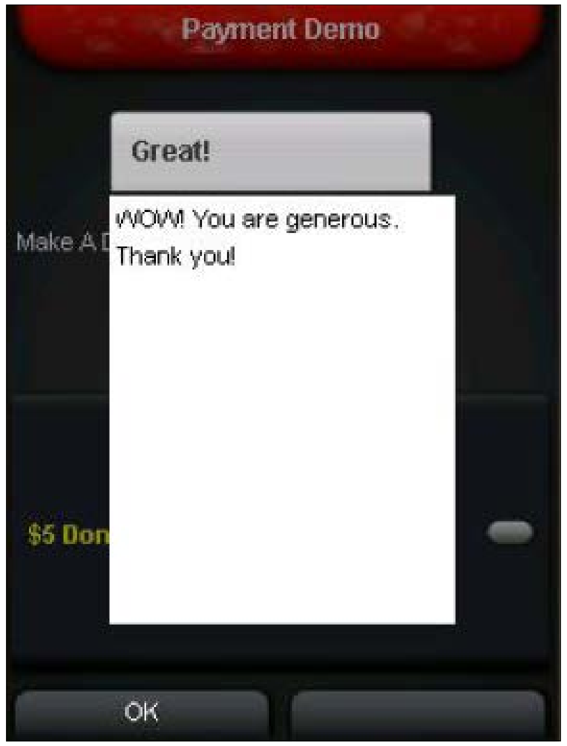
Are you sure, you want to buy this feature for EUR 5.00 using Premium Priced SMS provided by MyCompany?

Pay by: Premium Priced SMS - MyCompany

Update date: never

Update stamp: Thu Aug 12 23:30:00 GMT+10:00 2004

Figure 5


Payment Demo

Great!

WOW! You are generous. Thank you!

Figure 6

Conclusion

Although there are not many devices that implement the Payment API at this point, it is gaining popularity and is likely to be a popular choice for implementing payment options in the future. The Payment API makes the task of collecting funds from your target audience much easier because it leaves the final steps up to the device manufacturers.

This article introduced the Payment API as defined by JSR 229 and described how to define payment adapters, providers, and features within your JAD and JAR manifest files. Finally, it showed a simple example of this API in action using two toolkits with two different payment adapters. `</article>`

LEARN MORE

- JSR 229 specification
- Download the toolkits

MAKE THE FUTURE JAVA

Java™

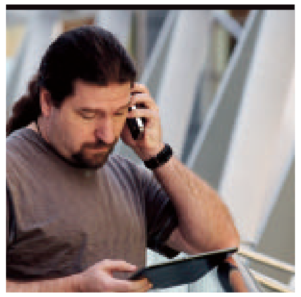
FIND YOUR JUG HERE

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

Hildeberto Mendonça
The Ceará Java User Group (CEJUG)

LEARN MORE

ORACLE®



TED NEWARD

BIO

Part 2

Oracle Berkeley DB Java Edition's Java API

Learn how to work with the base API of Oracle Berkeley DB.

In Part 1 of this series, we looked at some Oracle Berkeley DB basics.

Oracle Berkeley DB is a two-level data storage system that, when viewed from a really high level, looks vaguely relational on the surface, but it offers a much more "object-binary"-ish feel for managing data. Below that, there is a lower-level, more "key-value"-ish API called the *base API*, which offers even more flexibility in how data is managed and how data storage is customized.

We did not cover the base API in the last article, despite the fact that it is a lower-level API that offers some interesting solutions that other higher-level approaches would find difficult or impossible to offer.

IT'S SO EASY

Compared to other database systems, reading from and writing to the base API is absurdly simple.

Base API

Fundamentally, the base API of Oracle Berkeley DB models a key-value store, meaning each database—each database object, to be precise—is like one giant two-column table, the first column being a primary key column and the second column being a binary large object (BLOB) in relational terms.

In Oracle Berkeley DB terms, each key-value pair is a *record*, each of which is a DatabaseEntry class that is (internally) basically just a byte array.

I did mention that this is low-level access, right?

Opening and Closing a Database

Opening a database in the base API is pretty similar to creating an EntityStore from the Direct Persistence Layer (DPL), which is the layer we discussed in Part 1. We begin with the same Environment and EnvironmentConfig classes, only this time we use Environment and a DatabaseConfig object to open a Database object (as shown in **Listing 1**) instead of using a StoreConfig to yield an EntityStore object.

Once again, the DatabaseConfig object serves as the means by which we configure the behavior of the database. Of course, Newton's Law of Software ("What must be opened, must be closed") still holds, so we need to make sure we close everything:

```
@After public void close()
{
 db.close();
 dbEnv.close();
 dbDir.delete();
}
```

Remember, Oracle Berkeley DB will throw an exception if you fail to close these, so if an IllegalStateException: Unclosed Database: (your database name) is thrown from the Java Virtual Machine, you left one of these guys unclosed.

Temporary Databases

One of the uses to which Oracle Berkeley DB databases were frequently put in the UNIX world was as a temporary data store to hold in-flight data while the program was running, but serving no "persistence" purpose. In other words, the

PHOTOGRAPH BY PHIL SALTONSTALL

data was irrelevant and meaningless once the program quit, and in some cases, the data even created problems if it survived for the next run. (Think about a program that was reading input, calculating interim results, and storing them on disk to avoid having to hold all of them in memory, and then calculating more results and writing the final data somewhere else; those interim results written to disk would confuse the next run of the program.)

In order to help ensure that those unwanted bits don't survive past the run of the program, the Oracle Berkeley DB APIs offer the option to make the database temporary, meaning all the files representing the database are nuked when the program terminates. (To be quite precise about this, if the system can avoid writing to disk at all, it will, making these essentially in-memory databases. But the documentation specifically states that it is unsafe to assume that no disk I/O will occur. If all the data can't be maintained in cache, something will be written to disk, and then those files will be blown away later.) Because the system no longer has to worry about persisting data to disk, temporary databases are blazingly fast, though the obvious trade-off is the complete loss of data in the event of a crash.

To mark a database as a temporary database requires one call, setTemporary(true), on the DatabaseConfig object that is used to open the database. But beware: the disk underneath the configuration does matter, even if you never actually write to it. If, for example, you open an Environment on top of a directory in which a nontemporary database previously existed, Oracle Berkeley DB will throw another IllegalStateException claiming that the database can't be made temporary if it previously wasn't temporary.

Reading from and Writing to a Database

Compared to other database systems, reading from and writing to the base API is absurdly simple. Remember, it's a byte-array key-value store, so storing data to it consists of calling put with the key byte array and the value byte array, as shown in **Listing 2**.

I did mention that this is low-level access, right?

The first parameter for put is a Transaction object, which we can ignore (for now) because we want to keep things simple (and nontransactional). And, in the interest of reinforcing a point about the base API, the count() method on the Database object is a count of all the key-value pairs in the entire database, because, as I mentioned earlier, the database is essentially one giant collection of key-value pairs. Thus, we write one record and then assert to ensure that the database holds exactly one record.

The way to read a key-value pair from the database is pretty predictable (see **Listing 3**).

The first parameter to the get call is, again, a Transaction object, and the last is a LockMode enumeration that describes what kind of concurrency locks should be used—null states use the default settings given by the DatabaseConfig when the database was opened.

Removing a key-value pair from the database is equally predictable, as shown in **Listing 4**. And,

LISTING 1 / LISTING 2 / LISTING 3 / LISTING 4

```
File dbDir = new File("./data");
Environment dbEnv;
Database db;
@Before public void open()
{
 if (!dbDir.exists())
   dbDir.mkdir();

 EnvironmentConfig config = new EnvironmentConfig();
 config.setAllowCreate(true);
 dbEnv = new Environment(dbDir, config);

 DatabaseConfig dbConfig = new DatabaseConfig();
 dbConfig.setAllowCreate(true);
 db = dbEnv.openDatabase(null, "database", dbConfig);
}
```

Download all listings in this issue as text

with equal predictability, the first parameter to the delete call is, again, a Transaction object.

By the way, if you prefer to work with transactions, obtain one from the Environment and pass it in as the first parameter to any of these calls; this is a little bit beyond the scope of this article, but if you're familiar with transactions from relational databases, the Transaction object's API should be pretty easy to understand.

Deferred-Write Databases

One of the principal advantages the base API (and, in fact, Oracle Berkeley DB as a whole) offers is speed—for almost any operation done against an Oracle Berkeley DB database versus a traditional SQL database, the Oracle Berkeley DB API will be faster, though in many cases, things will be surrendered to get that speed (see our previous discussion of in-memory databases, for example).

One way that an Oracle Berkeley DB database can offer even higher performance is to defer the writes to disk to be done when convenient, as opposed to being done at the time of the put call. These are called *deferred writes*, and they offer a huge performance boost for systems whose data durability concerns aren't stringent.

Opening a database in deferred-write mode means calling setDeferredWrite(true) on the DatabaseConfig object before it is used to create the Database object. Once that is done, all writes to the database will be delayed until the developer calls sync() on the Database object to force a data write (which is also called a *checkpoint* or a *barrier* in some concurrency literature).

SUPERFAST
Because the system no longer has to worry about persisting data to disk, **temporary databases are blazingly fast**, though the obvious trade-off is the complete loss of data in the event of a crash.

The reason for the improved performance is simple: by deferring all the writes to disk until the developer issues the sync() call, Oracle Berkeley DB can do any updates to those just-written records in memory, rather than having to seek to disk and write them there, and then later it can write out only the final results of those updates.

Think, for example, of the benefits of deferred writes in a common end-user scenario: A user creates a record, edits the record, edits it some more, then realizes the record is unnecessary, and deletes the record. In a deferred-write scenario, when the developer finally calls sync(), nothing needs to be written, and the call can return immediately. The trade-off, of course, is that data could be lost in between sync() calls.

Note that temporary databases, as you might guess, are always writing in deferred mode.

Tuples and BIND

Storing things as a byte array is powerful but pretty awkward to work with. Sure, we could use Java's object serialization to transform every object into a byte array and back again, but serialization has the drawback of being pretty slow compared to alternatives (partly because it encodes type information into the serialized byte array, so not only must this information be read and written every time, it also bloats the storage requirements). So Oracle Berkeley DB offers another way to store (and transform) a collection of data elements: the TupleBinding and the BIND API.

Conceptually, a *tuple* is like a lightweight object: a collection of data elements that have no name associated with them (otherwise they'd be object fields). You can think of a tuple as an even lighter-weight data transfer object or parameter object.

In Oracle Berkeley DB, we can use a TupleBinding to describe how to put a variety of different elements into a record by creating a custom TupleBinding implementation to do the reads and writes (see **Listings 5a** and **5b**)—sort of like a highly specific Serializable that requires more work on the developer's part to implement.

This is a mouthful, but it's not as hard as it might look at first. We're looking to store RSSPost objects, which are essentially identical to the BlogPost objects from Part 1: a String title, a String of text, and a Date of posting. We want to store these RSSPost objects keyed by the title of the post. To do so, we need TupleBinding objects for both the key and the value.

The first is easy to obtain by asking TupleBinding for a TupleBinding instance for the "primitive" type String. The second is harder, because Oracle Berkeley DB has no built-in bindings for our RSSPost objects; thus, we have to create a binding by implementing the TupleBinding interface. (Normally, we would probably want to do this on the RSSPost object itself or as a standalone class, but to keep the example self-contained, I choose to do it as an anonymous inner class.)

Implementing that interface consists of two methods:

- One to take the object in question and write it to the TupleOutput object (for being written to the disk)
- And another to take the TupleInput (representing what's on the disk) and extract the fields—in the exact same order—into the RSSPost object to be returned

Once those two binding objects are in place, the rest is really just an exercise in get() and put(), as discussed earlier.

Searching

All this key-value stuff is great . . . if you already know the key of what you're looking for. When you don't, or when you want to get a range of things, it's not quite so helpful. Fortunately, Oracle Berkeley DB supports a Cursor API, giving you the opportunity to search through the records for the values you are looking for.

In theory, the Cursor API is much like a for loop: obtain a Cursor from the database, tell it to search based on a key, and then loop through looking for the key-value pairs you're searching for. (If you just start iterating, it will start with the first record in the database and give you all the records.) In practice, though, working with a Cursor is a bit trickier than it seems.

First, to make the sets a little easier to work through, let's configure the database to support duplicate records—by which we mean records with the same keys but different values all being stored successfully to the database. Doing so requires calling setSortedDuplicates(true) on the DatabaseConfig object before passing it in to obtain the Database instance.

Assuming that's done, putting a bunch of records into the database is straightforward, as shown in **Listing 6**.

Next, to obtain a Cursor, we ask the database to give us one, but then the Cursor needs to be told to search according to a key (which, again, must be in the form of a DatabaseEntry object). This key, as well as the DatabaseEntry that will hold the value, must be passed in, as shown in **Listing 7**.

The Cursor knows how many records fit the searched description, and it makes that information available to us through the count() method, which makes it easy to test whether it found all six of the records with the "Ted" key.

Iterating through the results, however, is a little tricky—when we call getNext() or getPrev(), the passed-in DatabaseEntry objects are populated with those values, but the first key-value pair was already fetched into cursorKey and cursorValue, so the loop looks a little weird (see **Listing 8**).

Like most of the other Oracle Berkeley DB objects, we have to make sure Cursor instances get closed. An IllegalStateException awaits those who fail to remember this.

Curiously enough, if we look in the resulting JUnit test text file (shown in **Listing 9**), we see some-

LISTING 5a / LISTING 5b / LISTING 6 / LISTING 7 / LISTING 8 / LISTING 9

```
@Test public void storeReadAndRemoveATuple()
  throws java.io.UnsupportedEncodingException
{
  RSSPost post = new RSSPost("The Vietnam of Computer Science",
                "Blah blah blah...");

  TupleBinding<String> stringBinding =
    TupleBinding.getPrimitiveBinding(String.class);

  TupleBinding<RSSPost> rssbinding =
new TupleBinding<RSSPost>() {
    public void objectToEntry(RSSPost post, TupleOutput to) {
      to.writeString(post.getTitle());
      to.writeString(post.getText());
      to.writeString(post.getPostingDate().toGMTString());
    }
    public RSSPost entryToObject(TupleInput ti) {
      RSSPost post = new RSSPost();
      post.setTitle(ti.readString());
      post.setText(ti.readString());
      post.setPostingDate(new Date(ti.readString()));
      return post;
    }
  };
```

Download all listings in this issue as text

thing interesting. Notice how all the values are now in lexicographical order. By default, Oracle Berkeley DB will sort records via straight byte-array value comparison, which, in this case, works well for us. If you ever want to change that sort order, you can write a custom Comparator and hand it to the database to use.

SERIOUS BOOST

Deferred writes offer a huge performance boost for systems whose data durability concerns aren't stringent.

Conclusion

Obviously, working with the base API is not quite as simple as working with the Direct Persistence Layer, but it definitely offers some lower-level flexibility in places where the DPL does not. In this case, we have the advantage that the same database can actually serve two needs—one as a key-value store (similar in some ways to Cassandra or BigTable) and one as an object storage database (similar to object-oriented database management systems such as db4o or Versant).

One way in which the base API might be preferable to the higher-level APIs is that this form is compatible with the original C-based Oracle Berkeley DB format. So, if a legacy C/C++ UNIX application (let's be honest—almost no Microsoft Windows apps ever knew Oracle Berkeley DB existed, much less were shipped using it) needs to be ported to Java, the base API offers a way to do that without requiring some kind of explicit conversion process in the code.

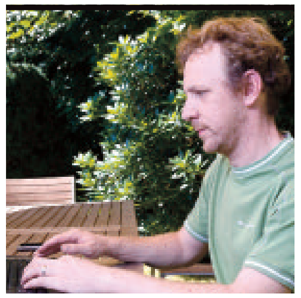
Despite all the time we've spent with Oracle Berkeley DB, there's still more to know, but we're going to have to put it down for now. Fortunately, the people working on it at Oracle have done a pretty good job documenting it, so if you're impatient to learn more, head on over to the Oracle Berkeley Database documentation page.

Regardless, knowing that Oracle Berkeley DB is there and available for use is the end goal. Now you know, and as the old 1980s cartoon show told us every week, "Knowing is half the battle!" </article>

LEARN MORE

- Oracle Berkeley DB

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



DataFX: Populate JavaFX Controls with Real-World Data

DataFX provides tools to retrieve, parse, and render data in a variety of JavaFX controls.

JOHAN VOS

BIO

PHOTOGRAPH BY TON HENDRIKS

The JavaFX 2 platform allows for a clean and attractive representation of data in different UI controls (for example, ListView, TableView, TreeView, Chart, and so on). The JavaFX 2 APIs provide a number of ways to render data, automatically visualize changes, and obtain user input. This is the classical *CRUD* approach that most applications require to enable users to create, read, update, and delete records in remote datasources, but in most applications, this approach requires a lot of boilerplate code.

It's fair to say that a developer of a domain-specific business application often wants to focus on the domain-specific logic—not on the messy code required to pull in relevant data—and then present the domain-specific functionality to users in the most useful way. In addition, the kind of boilerplate code needed for datasources is very easy to write incorrectly (and, thus, incur the wrath of hidden bugs). Code for retrieving, parsing, processing, and rendering data needs to be performant, thread-safe, and flexible.

Fortunately, this is where the open source BSD-licensed DataFX project steps in. DataFX facilitates the process of retrieving, parsing, and rendering data from different sources in different formats into different JavaFX controls.

DataFX contains two complementary major parts:

- **Datasources** provide tools to retrieve and parse the data.
- **Cell factories** are tools that make it easy to render data in a number of JavaFX controls.

Combining these two parts results in an end-to-end process for retrieving, parsing, and rendering the data into a simple-to-use yet very powerful API.

We'll now delve extensively into both areas.

Note: The full code for the example described in this article can be found here.

MAKING IT EASIER

DataFX facilitates the process of **retrieving, parsing, and rendering data** from different sources in different formats into different JavaFX controls.

Datasources

The first part of DataFX is focused on retrieving and parsing data. The second part concentrates on cell factories and facilitates the process of visualizing the parsed data to the controls; the second part will be covered later in this article.

The Java platform already contains a number of APIs that allow developers to obtain data from different sources, with a number of libraries providing convenient approaches for doing so.

There are two important characteristics of the JavaFX platform that should be considered when dealing with remote data:

- The Observable and ObservableList interfaces. These JavaFX interfaces allow for changes in data to immediately notify interested listeners, including UI controls. Instead of a single blocking get call, lists should be populated and data should be added when it is available. During this process, the user interface has to remain responsive. In fact, many UI controls automatically add listeners to the underlying datasource and automatically update the display when the data changes.

- Threading. All operations that modify the scene graph should be initiated from the JavaFX application thread. Hence, adding a data entry to an ObservableList should be done using the JavaFX application thread. On the other hand, data retrieval should not block the responsiveness of the user interface. The JavaFX Worker interface and the Service and Task classes provide interesting possibilities in this area.

Data that is rendered in JavaFX controls can have different origins, protocols, and characteristics. The DataFX API provides two concepts with a clear separation between them:

- The org.javafxdata.datasources.reader package contains the DataSourceReader interface and a number of implementations and builders.
- The org.javafxdata.datasources.provider package contains a number of DataSource provider interfaces, implementations, and builders.

The DataSourceReader abstracts the origin of the data (for example, a file, a network resource, or plain Java objects), while the DataSource abstracts the protocol of the data (for example, JavaScript Object Notation [JSON], XML, and Java objects) and of the destination (for example, Plain Old Java Objects [POJOs] for populating an ObservableList and String values for populating TableColumn instances).

The link between a DataSourceReader, the providers, and the controls is shown in **Figure 1**.

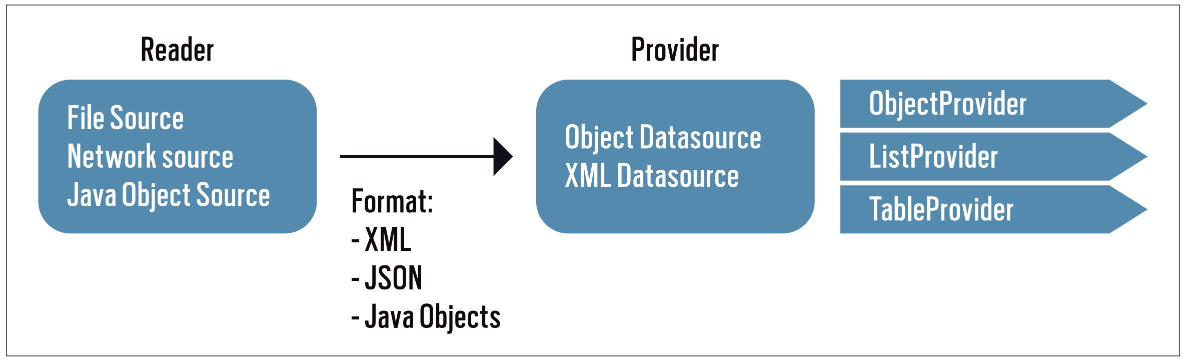


Figure 1

We will explain the flow by writing a small application that queries the Twitter API for tweets containing the term *JavaFX* and then renders those tweets in a ListView and a TableView.

DataSourceReader. DataSourceReader is an interface providing an InputStream that will be used by a DataSource. Different implementations of DataSourceReader will have their own specific mechanisms for obtaining data and creating the InputStream. The DataSource does not need to worry about the origin of the InputStream.

Currently, DataFX contains three implementations of DataSourceReader:

- FileSource provides data read from the file system.
- NetworkSource provides data read from the network.
- JavaObjectSource provides an ObjectInputStream containing Java objects.

These implementation classes can be used directly by developers.

DataFX contains a RestRequestBuilder class that allows developers to use the builder pattern to create a NetworkSource in a way that is familiar to Java API for RESTful Web Services (JAX-RS) developers. Indeed, many of today's business applications obtain their data from a Representational State Transfer (REST) service, in either JSON or XML. The RestRequestBuilder helps developers in making a call to a REST endpoint and in obtaining the resulting data.

The code in **Listing 1** creates a NetworkSource that will provide data obtained by calling the Twitter search API.

The RestRequestBuilder constructor takes the host name of the REST endpoint as a parameter. The complete path toward the endpoint is added by calling the path() method. This method can be called more than once, and the path segments will be added to each other. Query parameters can be

LISTING 1

```
RestRequestBuilder rrb = new RestRequestBuilder
("http://search.twitter.com").path("search.rss").queryParam
("q", "javafx");
DataSourceReader dataSourceReader = rrb.build();
```

Download all listings in this issue as text

specified using the queryParam() method, and form parameters can be specified using the formParam() method.

By default, a NetworkSource created with the RestRequestBuilder will use the GET operation, but the method() method can be used to specify a POST, PUT, or DELETE operation as well.

Calling the build() method on the RestRequestBuilder creates the NetworkSource. Note that at this stage, there is not yet a connection to the remote endpoint. The NetworkSource.getInputStream() method will initiate the connection to the REST endpoint.

The implementation classes in the org.javafxdata.datasources.provider package use the InputStream provided by the DataSourceReader.getInputStream() method. Those classes are completely agnostic with respect to where the InputStream originates from. This makes it easy to use, for example, a FileSource containing some XML or JSON data during testing and a NetworkSource containing similar XML or JSON data obtained via a REST call for production.

Creating a "test file" containing tweets can be done easily using a command-line tool such as Curl. In Linux, you would create a file containing tweets about JavaFX as shown in **Listing 2**.

CODE HELPER
Code for retrieving, parsing, processing, and rendering data needs to be **performant, thread-safe, and flexible.** Fortunately, this is where the open source BSD-licensed DataFX project steps in.

The file mytweets.xml can now be used as the input for a FileSource, as shown in **Listing 3**.

Although the DataSourceReader implementations are not dependent on the format of the data they deliver, they should be able to tell what format the data is in. Therefore, the constructors in the implementations allow for developers to specify the format of the expected response data, and the RestRequestBuilder contains a format() method that allows developers to specify the format (XML, JSON, or Java objects).

DataSource providers. The next steps in populating the JavaFX control are to parse the data and add tweets to the items in the ListView and the TableView. These steps will use the data obtained in the DataSourceReader. There is no difference between the two approaches we just discussed (a NetworkSource containing data obtained via a REST endpoint versus a FileSource containing static data on a file system).

The org.javafxdata.datasources.provider package contains a number of interfaces and a number of implementations and builders. There are a few criteria that determine which implementation class works best in a given context:

- Incoming protocol: The ObjectDataSource class supports XML, JSON, and serialized Java objects. Other implementations might support other protocols.
- Outgoing format: Some developers prefer to work with real Java objects, whereas others can live with XML. The ObjectDataSource converts incoming data into real Java objects. The XmlDataSource keeps an XML document about the incoming data and will provide the JavaFX controls with the required values by parsing the XML without converting it into a Java object.
- Consumer:
  - A DataSource that implements ObjectProvider provides general data that is observable.
  - A DataSource implementing ListProvider extends this ObjectProvider, and specifically provides an ObservableList of objects. Clearly, this is the appropriate interface when you want to populate a ListView.
  - Finally, a DataSource implementing TableProvider extends ListProvider and adds methods for setting and retrieving column information. This is what is needed for populating a TableView.

In order to create our Twitter example, we need an implementation that can handle XML or JSON (Twitter supports both). We will use the XML format in this example, but using JSON is equally easy.

The consumer of the data is either a ListView or a TableView. We will show how

LISTING 2 LISTING 3

```
curl "http://search.twitter.com/search.rss?q=javafx" >
/tmp/mytweets.xml
```

Download all listings in this issue as text

this can easily be accomplished using the ObjectDataSource. ObjectDataSource implements TableProvider—and thus also ListProvider—and can work with XML and JSON.

The ObjectDataSourceBuilder in the org.javafxdata.datasources.provider package is a convenience class that assists developers in creating an ObjectDataSource.

The code in **Listing 4** will create an ObjectDataSource and populate the ObservableList allTweets with the incoming tweets.

Note that we introduce a class named Tweet here, which is a POJO holding the data for a single tweet. Again, we are not required to use this POJO approach, but in case we want to use the Tweet concepts somewhere else in the application, a specific class is useful. The code for the Tweet class is shown in **Listing 5**.

After creating an ObjectDataSourceBuilder, we use the fluent API to give instructions about the ObjectDataSource we want to create:

- The parsed entries should be converted to instances of Tweet.
- The data should be obtained from the DataSourceReader we just created.
- Incoming tweets are in XML elements named item.
- The resulting entries should be added to an existing ObservableList named allTweets.
- When the retrieved entities should be rendered by a TableView, we are interested only in the author and title parts of the tweet, not in the publication date.

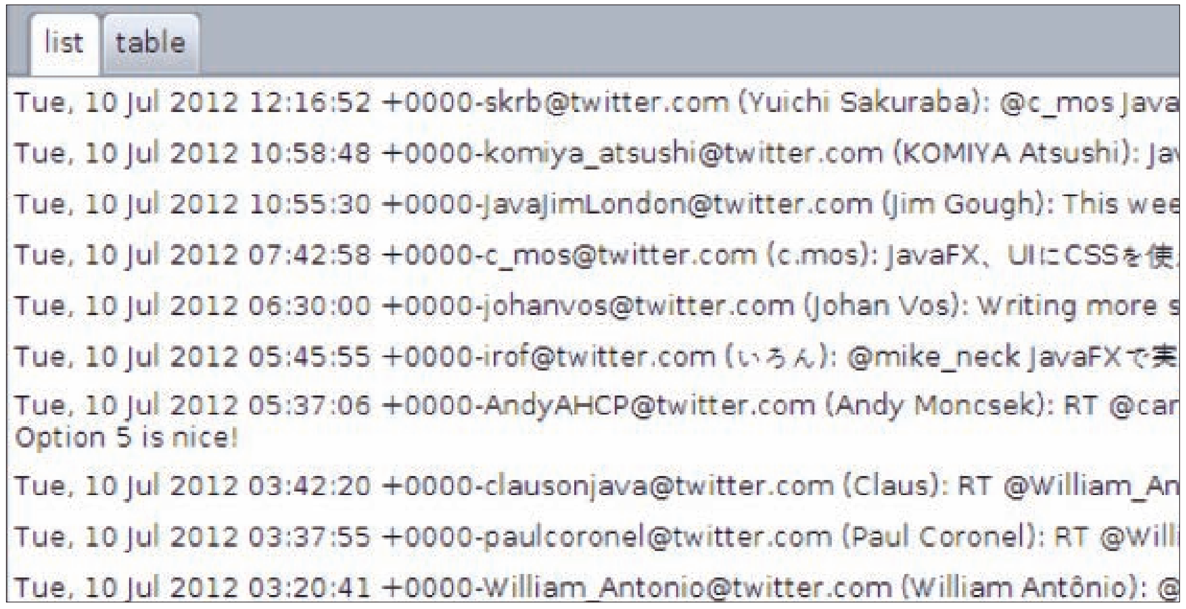
Next, the ObjectDataSource is created. Data is not retrieved until the DataSource.retrieve() method is called. All implementations in the org.javafxdata.datasources.provider classes will immediately return when this method is called. Retrieving and parsing data is handled in a background task using the JavaFX Service concept. Therefore, it is important that the retrieve() method is called from the JavaFX application thread.

When data is obtained and parsed into entries, those entries are added to the desired ObservableList. If an existing ObservableList is supplied using the ObjectDataSourceBuilder, it will be used. Otherwise, a new, initially empty ObservableList will be created and populated as soon as entries are parsed. The resulting ObservableList can be obtained at any point using the getData() method on the ObjectDataSource. A DataSource also contains a runningProperty, which is a convenience property for indicating whether this DataSource is still

LISTING 4 LISTING 5

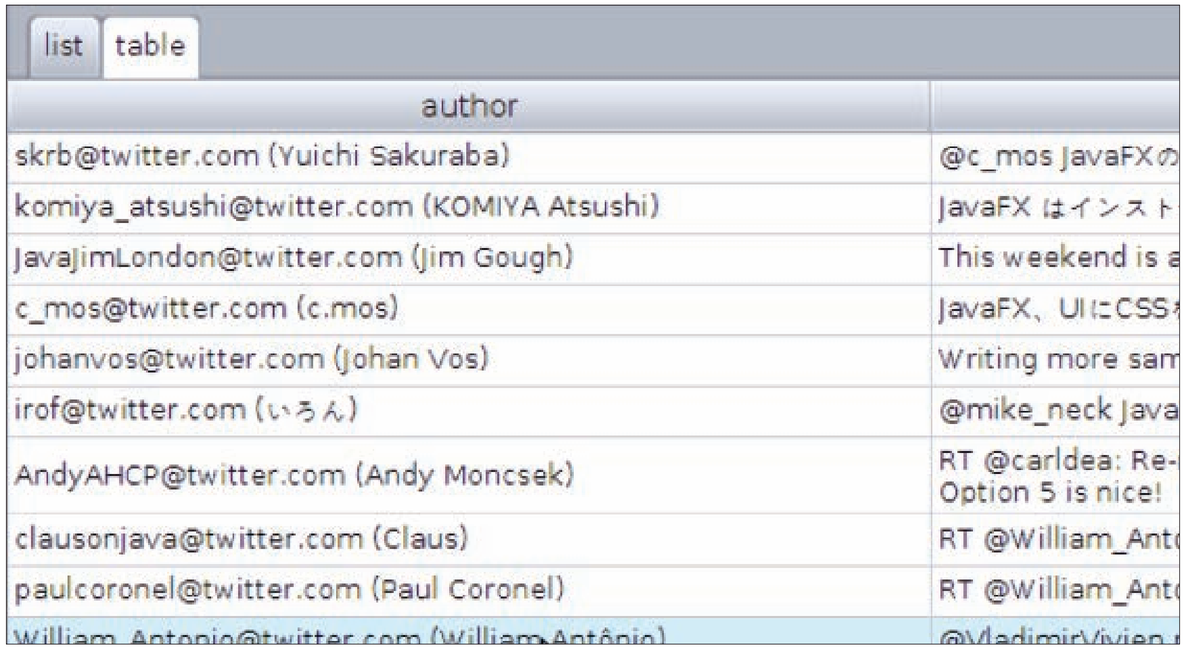
```
ObjectDataSource createDataSource() {
  RestRequestBuilder rrb =
new RestRequestBuilder("http://search.twitter.com")
        .path("search.rss")
        .queryParam("q", "javafx");
  DataSourceReader dataSourceReader = rrb.build();
  ObjectDataSourceBuilder<Tweet> builder =
        ObjectDataSourceBuilder.<Tweet>create();
  builder.itemClass(Tweet.class)
        .dataSourceReader(dataSourceReader)
        .itemTag("item")
        .resultList(allTweets)
        .columns("author", "title");
  ObjectDataSource dataSource = builder.build();
  return dataSource;
 }
```

Download all listings in this issue as text


list table

Tue, 10 Jul 2012 12:16:52 +0000-skrb@twitter.com (Yuichi Sakuraba): @c_mos Java
Tue, 10 Jul 2012 10:58:48 +0000-komiya_atsushi@twitter.com (KOMIYA Atsushi): Ja
Tue, 10 Jul 2012 10:55:30 +0000-JavaJimLondon@twitter.com (Jim Gough): This wee
Tue, 10 Jul 2012 07:42:58 +0000-c_mos@twitter.com (c.mos): JavaFX、UIにCSSを使
Tue, 10 Jul 2012 06:30:00 +0000-johanvos@twitter.com (Johan Vos): Writing more s
Tue, 10 Jul 2012 05:45:55 +0000-irof@twitter.com (いろん): @mike_neck JavaFXで実
Tue, 10 Jul 2012 05:37:06 +0000-AndyAHCP@twitter.com (Andy Moncsek): RT @car
Option 5 is nice!
Tue, 10 Jul 2012 03:42:20 +0000-clausonjava@twitter.com (Claus): RT @William_An
Tue, 10 Jul 2012 03:37:55 +0000-paulcoronel@twitter.com (Paul Coronel): RT @Willi
Tue, 10 Jul 2012 03:20:41 +0000-William_Antonio@twitter.com (William Antônio): @

Figure 2


list table

author	
skrb@twitter.com (Yuichi Sakuraba)	@c_mos JavaFXの
komiya_atsushi@twitter.com (KOMIYA Atsushi)	JavaFX はインスト
JavaJimLondon@twitter.com (Jim Gough)	This weekend is a
c_mos@twitter.com (c.mos)	JavaFX、UIにCSS
johanvos@twitter.com (Johan Vos)	Writing more sam
irof@twitter.com (いろん)	@mike_neck Java
AndyAHCP@twitter.com (Andy Moncsek)	RT @carldea: Re- Option 5 is nice!
clausonjava@twitter.com (Claus)	RT @William_Ant
paulcoronel@twitter.com (Paul Coronel)	RT @William_Ant
William_Antonio@twitter.com (William Antônio)	@VladimirVivien

Figure 3

LISTING 6 LISTING 7 LISTING 8

```
void buildListTab(Tab t) {
  ListView listView = new ListView(allTweets);
  t.setContent(listView);
}
```

Download all listings in this issue as text

actively processing data.

Note that calling the retrieve() method on ObjectDataSource returns the background Service object that is reading and parsing the entries. As a consequence, progress on this process can also be followed and queried using the existing properties on Service (that is, stateProperty).

The same ObjectDataSource can be used for a ListView and a TableView. Our example shows a TabPane with two tabs: one containing a ListView and the other containing a TableView.

The relevant code for populating the ListView is now very simple (see **Listing 6**). The allTweets field is the ObservableList instance that is filled by the ObjectDataSource.

Populating the TableView and defining which columns to show is done with the code shown in **Listing 7**. Apart from populating the TableView with the allTweets list, we also populate the columns with the TableColumn instances returned by the datasource.

The results of the combined code from **Listing 6** and **Listing 7** are shown in **Figure 2** and **Figure 3**.

Cell Factories

The JavaFX controls allow for a clear separation between the physical parts of the control (for example, a Cell in a ListView or a TableView) and the content rendered in the control. The "glue" between the content and the representation of the content in the cell is provided by factories.

A ListView has a cellFactory property, and a TableView has a List of TableColumn instances that each has a cellFactory property as well. Alternatively, a TableView also has a rowFactory property.

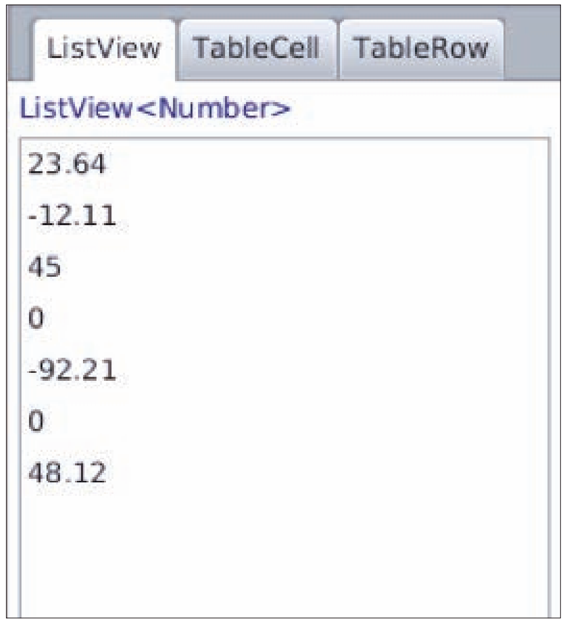
Let's look at the ListView first. The cellFactoryProperty on the ListView is obtained as shown in **Listing 8**.

The Callback interface declares a single method:

```
public interface Callback<P,R>
R call (P param)
```

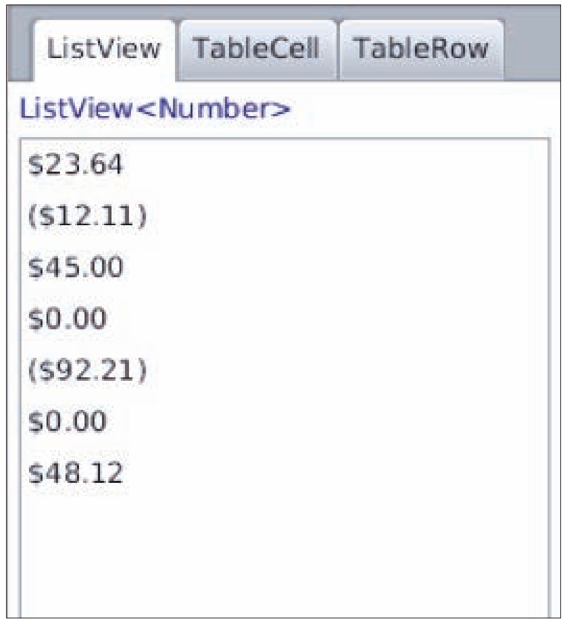
A custom cellFactory, therefore, has to provide a Callback implementation, with a call method accepting a ListView<T> parameter and returning a ListCell<T> value:

```
ListCell<T> call (ListView<T>)
```



```
23.64
-12.11
45
0
-92.21
0
48.12
```

Figure 4



```
$23.64
($12.11)
$45.00
$0.00
($92.21)
$0.00
$48.12
```

Figure 5

Using TableRow instances in a TableView, the situation is similar. The rowFactoryProperty on TableView is obtained as shown in **Listing 9**.

If we want to provide a rowFactory, we have to create a Callback implementation with a call method that is declared as follows:

```
TableRow<S> call (TableView<S>)
```

However, rowFactory is rarely used. Instead, the preferred approach on TableView is to install cell factories into each TableColumn instance. As the name states, a TableColumn is responsible for managing the contents of a single column in a TableView.

Each TableColumn contains a cellFactory property that is obtained as shown in **Listing 10**.

If we want to provide a custom implementation for a cellFactory for a particular column in a table, we have to create an implementation of a Callback class defining the following call method:

```
TableCell<S,T> call
(TableColumn<S,T>)
```

Although all of this is doable, it frequently tends to require a relatively high amount of boilerplate code compared to the desired outcome.

Therefore, DataFX contains a number of predefined cell factories. Some of the cell factories that are currently in JavaFX 2.2 were

LISTING 9 / LISTING 10 / LISTING 11 / LISTING 12

```
public final
ObjectProperty<Callback<TableView<S>,TableRow<S>>>
rowFactoryProperty
```

Download all listings in this issue as text

first started in DataFX and moved to the JavaFX 2 repository later. The DataFX repository contains useful but experimental cell factories that have not yet made it into the JavaFX 2 repository.

Cell factory example. As an example, we show the MoneyCellFactory here. Consider the code in **Listing 11**. This simple example will create a ListView without specifying a cellFactory. The resulting ListView will look like **Figure 4**.

If we want the numbers in this list to represent money, we can leverage the MoneyCellFactory by adding the single line shown in **Listing 12**. The output now looks like **Figure 5**.

Like all cell factories in DataFX, the MoneyCellFactory provides methods for creating cell factories for ListView as well as for TableColumn instances.

Developers don't have to worry about creating implementations of the Callback interface, making sure they have the right parameters in the correct order, and so on.

Conclusion

We have seen how DataFX provides tools that enable developers to effectively retrieve, parse, and render data in a variety of JavaFX controls with code that is performant, thread-safe, and flexible.

DataFX has two complementary major parts:

- Datasources provide tools to retrieve and parse the data.
- Cell factories are tools that make it easy to render data in a number of JavaFX controls.

The lives of JavaFX developers should get a little bit easier as they learn to deploy these tools. </article>

LEARN MORE

- DataFX
- The DataFX project
- DataFX source code repository

THE FUTURE OF GRAAL

Oracle Labs' **Dr. Thomas Wuerthinger** explains the goals of the Graal Project.

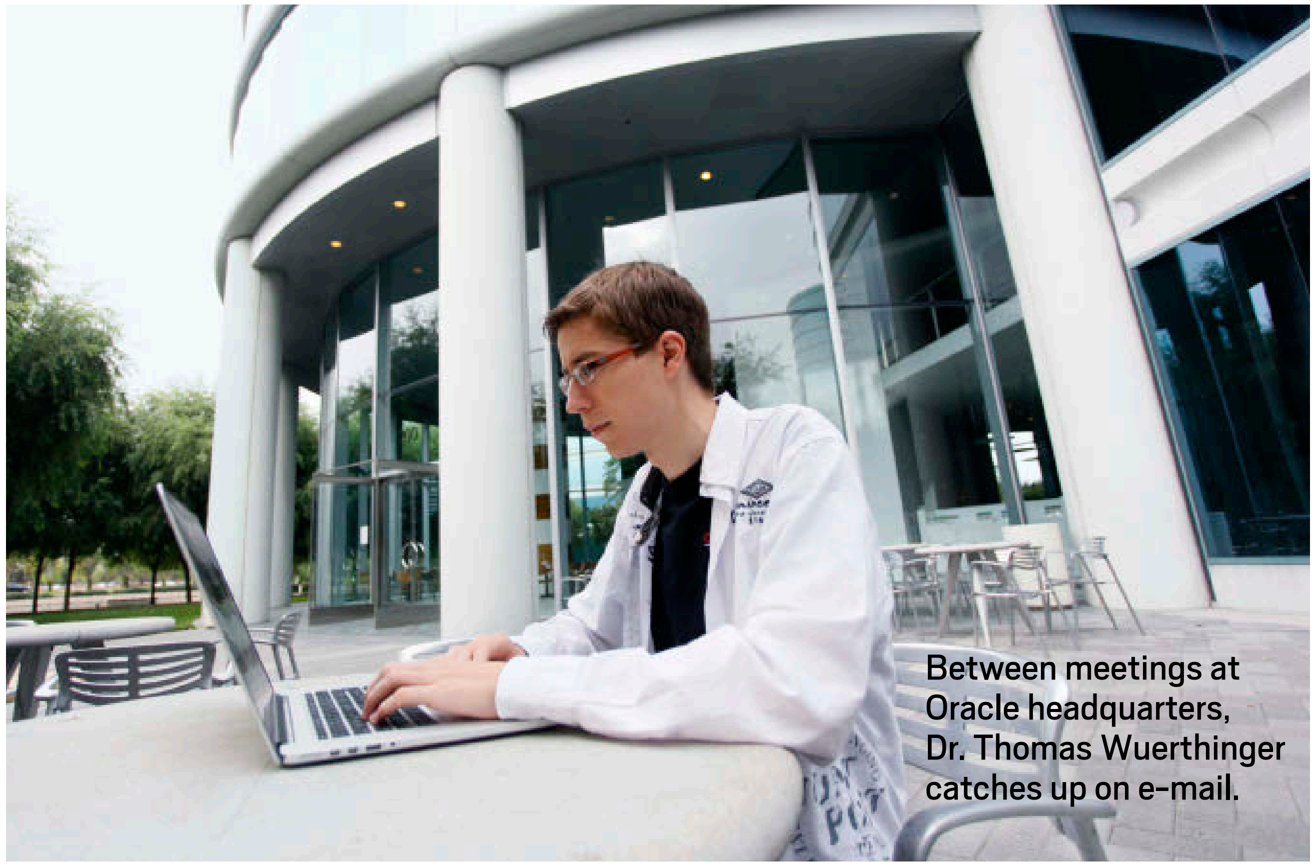
BY JANICE J. HEISS

PHOTOGRAPHY BY BOB ADLER

The Graal Project, which is run by Oracle Labs, aspires to implement a dynamic compiler in Java that produces excellent code quality without compromising compile time and memory usage in the Java Virtual Machine (JVM). The result could be a seamless integration between application and virtual machine (VM). The project is run by Thomas Wuerthinger, who is senior member of the technical staff at Oracle Labs. Wuerthinger received his PhD in computer science in 2011 from Johannes Kepler University in Linz, Austria. Previously, he worked on the IdealGraphVisualizer, the Crankshaft/V8 optimizing compiler, and the Dynamic Code Evolution VM. We met with him to get an update on developments in Graal.

Java Magazine: What does the Graal Project hope to accomplish?
Wuerthinger: In brief, the project is "a quest for the JVM to leverage its own J." We want to create a VM in which major components are written in Java so that they can benefit from Java's safe execution model and tooling. Also, the modular design of the Graal VM aims at improving extensibility and maintainability.
Java Magazine: How far along are you in reaching your goal?
Wuerthinger: A main focus of the project so far has been to develop interfaces for the Graal VM so that it can be easily extended. This should encourage people to experiment with the VM and in that way contribute to our vision. Additionally, we have developed a prototype of a dynamic compiler for Java that provides a peak performance that lies between the client and the server configuration of the HotSpot VM. Like the Jikes RVM [Research Virtual Machine] or the Maxine VM, our system shows that it is possible to write system software using the Java language.
Java Magazine: It's argued that the Graal compiler will allow Java libraries to extend their functionality in ways not currently possible, so that more new languages can be efficiently implemented. Can you give us some details about this? For example, which new languages will be implemented?
Wuerthinger: Supporting multiple languages is a main goal of Graal. We want to provide the capabilities of the Java platform for all languages. In particular, we developed a multilanguage framework that allows language implementers to focus on implementing the semantics of their language without worrying about compiler or VM details. The semantics of the language are specified by a Java program that implements an AST [abstract syntax tree] interpreter for that language. We developed a technique that enables the Graal compiler to deliver high-performance execution for languages implemented in that framework. We developed prototypes for a few languages in order to shape the API, and we invite language implementers to give this framework a try.
Java Magazine: Why should the average Java developer, looking ahead a few years, care about Graal? What might it enable them to do that they cannot do now?
Wuerthinger: The Graal project is more experimental than many other OpenJDK projects. Therefore, we cannot make promises that what we do will eventually be shipped as a product. Still, we believe that the Graal project will be a positive contribution and

SUPPORT FOR ALL
"Supporting multiple languages is a main goal of Graal. We want to provide the capabilities of the Java platform for all languages."


Between meetings at Oracle headquarters, Dr. Thomas Wuerthinger catches up on e-mail.

Wuerthinger (third from right) with several of the PhD students who are contributing to the Graal project at Johannes Kepler University in Linz, Austria

PHOTOGRAPH COURTESY OF JOHANNES KEPLER UNIVERSITY

help keep the Java ecosystem vibrant over the next few years. In particular, we expect Graal to help us learn more about how to leverage the Java language within the JVM. This could lead to both performance improvements for Java itself as well as better support for additional languages.

Also, the flexibility of Graal should enable its application in a wide range of different domains including cloud infrastructure, heterogeneous computing environments, and embedded devices. We want to make sure that Java continues to stand up to its slogan, "Write once, run everywhere."

Java Magazine: What should developers who are interested in Graal do?

Wuerthinger: We want to keep the entry barrier to Graal development as low as possible. Therefore, we have placed a small description on how to build and run Graal on the front page of our project. Also, we encourage developers to join our OpenJDK mailing list. Once a week, the changes to the Graal code base are posted to that list.

Java Magazine: The Graal project claims as one of its goals leveraging Java within the VM. What are some of the specific advantages and challenges involved in writing parts of the JVM in Java?

Wuerthinger: A main advantage is improved robustness by executing the compiler in a managed environment. This means that an error in the compiler could, for example, lead to a null pointer exception instead of crashing the VM process. Also, we can make use of Java reflection and annotations. There is no language barrier between the compiler and the application, making it a lot easier for Java applications to influence or extend the compiler. There are challenges in two areas: We must make sure our compiler is memory efficient and does not degrade garbage collection performance. Also, the compiler has to be precompiled in order to enable a fast startup.

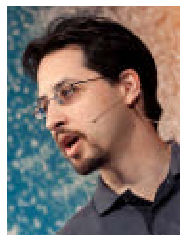
Java Magazine: What appeals to you about working with Graal?

Wuerthinger: I like working on system software while still having all the tooling and benefits of programming in Java. Also, I enjoy the ability to do free-spirited experiments on the JVM. The experimental nature of the project allows us to try new approaches without being immediately restricted by practical barriers. Additionally, I like working on the Graal team, which consists of a mix between experienced developers and curious master's and PhD students! `</article>`

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

//fix this /

Hint: The builder syntax is great, but sometimes it can hide the obvious.



In the July/August 2012 issue, Jason Hunter and Boris Shukhat presented us with a connection pooling code challenge. They gave us a trimmed code snippet from the pool class and asked us to fix it without a major redesign.

The correct answer is #4. The ConnectionPool.java code erroneously assumed that hashCode() and equals() would not be overridden. The upgrade to a new driver broke the program because the new implementation implemented equals() using object equivalence instead of the default reference equality. The approach in #4 allows us to make our program absolutely independent from implementations of hashCode() and equals(). The solution is to create a safe implementation of the Connection interface wrapping the real Connection implementation and routing calls to it internally so that the program will always use our object's reliable hashCode() and equals() methods.

This issue's challenge comes from Stephen Chin, a Java Evangelist at Oracle and coauthor of *Pro JavaFX 2* (Apress, 2012).

1 THE PROBLEM

The JavaFX Media APIs make it very easy to add video content to your desktop application. While the APIs are quite rich, this also means that there are several classes involved, so it can be easy to get simple use cases wrong if you are not careful.

2 THE CODE

In this code snippet from the Application start method of the MediaQuiz class, can you identify why the video clip does not play?

```
stage.setScene(SceneBuilder.create()
  .width(960).height(540)
  .root(
    StackPaneBuilder.create().children(
      new MediaView(
        new MediaPlayer(
          new Media(
            getClass().getResource("quiz.mp4").toString()))))
    .build())
  .build());
stage.show();
```

I used the following source layout:

- org
  - steveonjava
    - MediaQuiz.java
    - quiz.mp4

3 WHAT'S THE FIX?

1) The resource path is wrong for loading the media file.

2) JavaFX 2 does not support H.264 encoded video.

3) There is a missing control function to enable the video.

4) The MediaView won't be visible without the proper StackPane layout constraints set.

GOT THE ANSWER?

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

ART BY I-HUA CHEN

EDITORIAL
Editor in Chief
Caroline Kvitka
Community Editors
Cassandra Clark, Sonya Barry, Yolande Poirier
Java in Action Editor
Michelle Kovac
Technology Editors
Janice Heiss, Tori Wieldt
Contributing Writer
Kevin Farnham
Contributing Editors
Blair Campbell, Claire Breen, Karen Perkins

DESIGN
Senior Creative Director
Francisco G Delgadillo
Senior Design Director
Suemi Lam
Design Director
Richard Merchán
Contributing Designers
Jaime Ferrand, Nicholas Pavkovic
Production Designers
Sheila Brennan, Kathy Cygnarowicz

PUBLISHING
Publisher
Jeff Spicer
Production Director and Associate Publisher
Jennifer Hamilton +1.650.506.3794
Senior Manager, Audience Development and Operations
Karin Kinnear +1.650.506.1985

ADVERTISING SALES
Associate Publisher
Kyle Walkenhorst +1.323.340.8585
Northwest and Central U.S.
Tom Cometa +1.510.339.2403
Southwest U.S. and LAD
Shaun Mehr +1.949.923.1660
Northeast U.S. and EMEA/APAC
Mark Makinney +1.805.709.4745
Recruitment Advertising
Tim Matteson +1 310-836-4064
Mailing-List Rentals
Contact your sales representative.

RESOURCES
Oracle Products
+1.800.367.8674 (U.S./Canada)
Oracle Services
+1.888.283.0591 (U.S.)
Oracle Press Books
oraclepressbooks.com

ARTICLE SUBMISSION
If you are interested in submitting an article, please e-mail the editors.

SUBSCRIPTION INFORMATION
Subscriptions are complimentary for qualified individuals who complete the subscription form.

MAGAZINE CUSTOMER SERVICE
java@halldata.com **Phone** +1.847.763.9635

PRIVACY
Oracle Publishing allows sharing of its mailing list with selected third parties. If you prefer that your mailing address or e-mail address not be included in this program, contact Customer Service.

Java Magazine is published bimonthly with a free subscription price by Oracle, 500 Oracle Parkway, MS OPL-3C, Redwood City, CA 94065-1600.

Digital Publishing by Texterity

Java Skills #1 Priority Of Hiring Managers

Get Certified with Oracle University

- ✓ Prepare with Java experts
- ✓ In the classroom or online
- ✓ Pass or retest for free
- ✓ And save up to 20%

Java™ Click to Register

ORACLE®

Source: from Dice.com's *"Dice Report"*; *"January 2012: The Top Spots"*