Java™ magazine
B the Java community

//MARCH/APRIL 2012 /

The New Java Developers

Meet some of the young developers creating the future of Java 11

//table of contents /

COVER ART BY VARIOUS PHOTOGRAPHERS, SEE PAGE 11
VISUAL EFFECTS BY BOB WASILOWSKI, ENRIQUE AGUIRRE, I-HUA CHEN

It's certainly a pleasure to draw attention to the new Java developers around the world. Until the past year or so, few would argue that the Java ecosystem had stagnated. The anti-Java forces had their knives out, and the haters were doing what they do best.

Things have changed now. Java SE 7 is available, and Java SE 8 is on the way; Java developer conferences around the world are selling out in short order; Java skills are in high demand by recruiters; and the Java community is reinvigorated thanks to efforts including the OpenJDK project, the Adopt-a-JSR program, and—if I may be so bold—even this publication. Furthermore, we are in the midst of a Golden Age of Programming Languages, in which the most-exciting candidates all target the Java Virtual Machine, as well as a Golden Age of Web-scale Applications, for which Java technology is turning out to be the path to success (just ask Twitter).

As one speaker at the recent Jfokus conference in Sweden remarked, "Java isn't dying—it has grown up and is giving birth to its children." I couldn't have come up with a more apt statement as a theme for this issue.

In this issue's cover story, "The New Java Developers," Tori Wieldt profiles a small selection of young (ages 18 to 27) Java developers from around the world. The story is truly inspirational. These young people span cultural, geographical, and intellectual divides, but they all have one thing in common: a passion for Java technology, the same passion that we all saw in other developers their age back in the 1990s. The baton has been passed—and the community can take comfort in its own renewal.

Justin Kestelyn, Editor in Chief BIO

//send us your feedback /

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

PHOTOGRAPH BY BOB ADLER

Girls immersed themselves in technology at Dare 2B Digital, where they got creative with Alice and heard from successful women in technology.

Staci Lyons

GIRL POWER

One hundred girls learned Java with Alice software in a February 11 workshop at Dare 2B Digital 2012, an annual conference that encourages girls ages 14 and up to pursue technical careers. "Young women use technology but don't associate themselves with creating it," says **Ruth Sergiou**, organizer of the conference. In this Oracle Academy workshop, students created a 3-D animation and learned Java basics with the help of Oracle volunteers. The Oracle Women's Leadership program sponsored the workshop. **Staci Lyons**, vice president of global business development at Oracle, talked about her career path and shared her enthusiasm for technology in a kickoff panel.

PHOTOGRAPHY BY MARTIN STEIN, ART BY I-HUA CHEN

Java Olympics Under Way

Round Two of the 2011–2012 Java Olympics is currently under way in 10 cities in Belarus, Kazakhstan, Russia, Ukraine, and Uzbekistan. More than 1,500 students from 322 universities participated in Round One, which took place in December. In that round, contestants completed a 40-question, 45-minute online quiz. **Alexander Belokrylov**, of Oracle St. Petersburg, reports that 75 percent correct was the highest grade, with 55 percent being the average grade. "Next time we should decrease the complexity of the quiz," Belokrylov concludes. The Round One winners were invited to travel to 1 of 10 cities across the region to participate in Round Two, which began in February and continues through March. Each student is provided with a PC preloaded with JDK 7, NetBeans 7.0.1, and Java 7 API documentation. The contestants have four hours to solve seven tasks by writing Java applications. The contestants who complete the most tasks in the least time will be the winners in each city.

The final round will take place in May in Astana, Kazakhstan. In addition to receiving awards after the close of Round Three, the final winners will be invited to travel to Moscow in 2013 to attend JavaOne Russia April 12–13.

Learning and Teaching with Minecraft

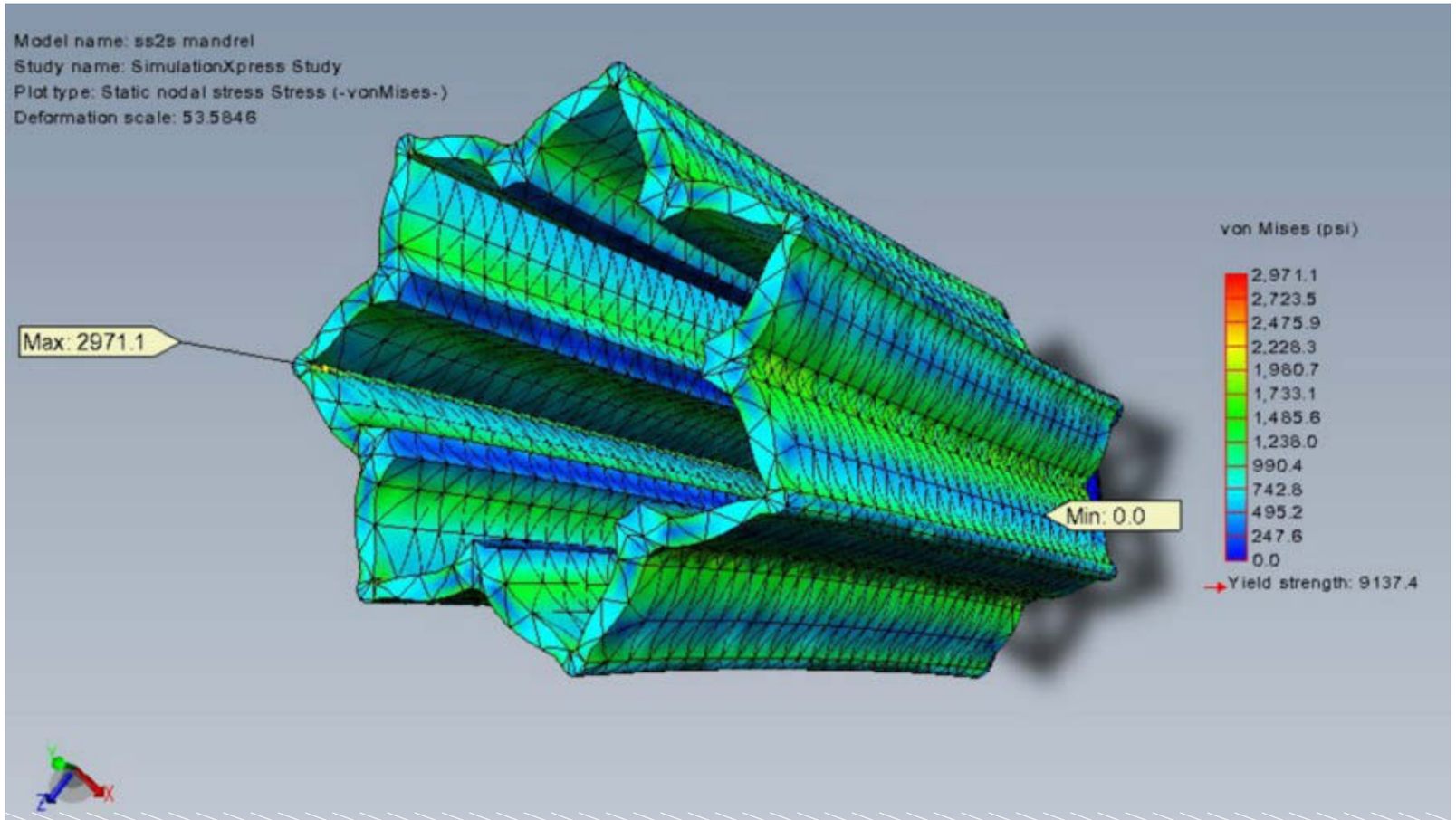
Students in middle school and high school are learning Java to modify Minecraft. The game-builder software, written in Java, uses simple colored blocks to create any virtual environment. The game has huge appeal, with 22 million registered users—nearly 5 million of whom have bought the game. Students learn Java to build "mods"—additional visual components that improve their virtual environment and characters. Students are so engaged with Minecraft that teachers are using it to teach math, physics, history, and English. "It is so open ended and modifiable," explains **Joel Levin**, who teaches computer classes to primary and high school students in New York, New York (and is known as *the* Minecraft teacher). "I was able to reshape it into a teaching tool with structures and instructions for students to follow." In collaboration with **Markus Persson**, the Swedish creator of Minecraft, Levin offers educational resources and a discount license for schools on his Website.

Minecraft is first and foremost a video game, but parents need not knock it. Teachers are also using it to engage students in math, physics, English, and history. And these students are learning Java, too.

PHOTOGRAPHS COURTESY OF MINDCRAFTEDU.COM, WINTRISS TECHNICAL SCHOOL

Java Changes Lives

At Wintriss Technical School, the motto is "Changing Kids' Lives with Java Computer Programming." The San Diego, California, school offers Java programming classes with the NetBeans IDE to primary and middle school students. Java professionals from the San Diego Java User Group teach the after-school and weekend classes and share their love for programming with students, motivating them to get into it themselves. Four students, in grades 8 to 10, have passed the Advanced Placement Computer Science exam. The school also organizes an annual International Autonomous Robot Competition to be held June 23–24, 2012.



JAVA AND SUGAR POWER ROCKETS

As part of the Sugar Shot to Space project, a team of engineers, chemists, mathematicians, and experts in amateur rocketry from all over the world are preparing a rocket powered by a sugar propellant to be sent 100 kilometers into space. For this project, Oracle Labs donated Sun SPOTs (Small Programmable Object Technology), the Java development toolkit, to test and remotely control the rockets. "Analysis is very important in detecting issues in engineering and especially for a spaceship," says **John Resler**, a software engineer working on the project. "This is precisely what the SPOT technology has been designed for." The goal of the project is to improve the performance of engines using sugar propellant.

IMAGE COURTESY OF SUGAR SHOT TO SPACE

Global JUG Growth Continues

Among the signs that Java is indeed moving forward are the flourishing new Java user groups (JUGs) that have been springing up across the world. For example, in the fall of 2010, **Csaba Toth** and **Greg Turnquist** started the Nashville Java User's Group in Nashville, Tennessee. Two people showed up for the first meeting in October 2010; today, more than 25 developers regularly attend NJUG's monthly meetings.

Other JUGs that recently celebrated their first anniversaries include the New Hampshire Java Users Group, founded by **Ted Pennings**, **Matt Merrill**, and **Scott Curry** in October 2010; JUG Ostrava in the Czech Republic, founded by **Ondřej Kvasnovský** in November 2010; and CaJUG in Cariri, Ceará, Brazil, founded by **Alday Pinheiro** in December 2010.

Very young JUGs include the Algeria Java User Group, founded by **Ferhat Guezlane** in September 2011; FinistJUG in Brest, France, founded by **Mikael Le Berre** and **Horacio Gonzalez** in December 2011; and the Pakistan Java User Group, founded by **Shahzad Badar** in December 2011. Congratulations and good luck to these new JUGs—and to all that are soon to be founded.

From top left: Duke welcomes visitors to Oracle headquarters; conference participants gather outside the Oracle conference center; Oracle's Sharat Chander

In this Java Spotlight podcast recorded at the conference, JUG leaders discuss how to increase member involvement and other challenges.

USER GROUP SUMMIT

The annual International Oracle Users Group Community (IOUC) Leader's Conference, held January 23–25 at Oracle headquarters in Redwood Shores, California, focused on fostering communication between user group leaders and Oracle. The Leader's Conference included three days of learning, networking, sharing, and collaborating about user groups and Oracle products and technologies. The summit included user groups focused on Java, MySQL, applications, database, technology, and industries.

In the dedicated Java track, Java user group (JUG) leaders from around the world got updates from the Oracle Java development team, discussed the status of programs such as Adopt-a-JSR, shared best practices, and started collaborating on future projects.

PHOTOGRAPHS BY YOSHIO TERADA

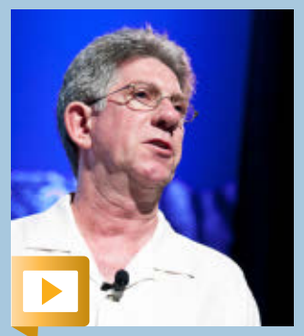
Developers Foresee a Profitable 2012

The results of several Java.net polls spanning the end of 2011 and the start of 2012 suggest that Java and Java Virtual Machine (JVM) developers are optimistic regarding both Java's continuing progress and the Java/JVM job market. Reflecting on 2011, developers considered the Java 7 release to be the most important Java/JVM–related happening, with the growth of Android a distant second. As for 2012, developers did not reach a consensus on what the most important Java/JVM news will be. Advances are expected on all fronts, from Java on smartphones and the OpenJDK to Java EE and the cloud.

While developers don't foresee a predominant Java/JVM–related news story arising in 2012, 67 percent of respondents in the Java.net job opportunities poll expect opportunities for Java/JVM developers to remain stable or increase in 2012.

JCP.next Evolution Continues

Building on the success of the now-complete JSR 348 (Towards a New Version of the Java Community Process), JSR 355 has been submitted. The core objective of JSR 355 is to merge the two current Java Community Process (JCP) Executive Committees (SE/EE and ME) into a single JCP Executive Committee on the grounds that Java is one platform.



Specification Lead Patrick Curran discussed JSR 355 in a recent Java Spotlight podcast.

In addition to addressing the merger of the executive committees, JSR 355 includes consideration of important rule changes, such as how many "yes" votes will be required for a JSR to be approved. You can participate in the decision-making process via the public JSR 355 project. There you'll find the freely accessible JSR 355 Document Archive, message forum, and JIRA issue tracker.

PHOTOGRAPHS BY PINAR OZGER, GETTY IMAGES

EVENTS

JAVAONE TOKYO *APRIL 4–5, TOKYO, JAPAN*

JavaOne, the world's best conference for the Java community, is headed for Japan. JavaOne offers Java experts and enthusiasts two days of learning and networking focused entirely on all things Java. Sessions and demos cover everything from building modern applications using JDK 7 to leveraging JavaFX, creating client-side applications, and more. Connect with some of the world's foremost Java experts, collaborate with peers, and celebrate your role in the vibrant and growing Java community. Register today!

APRIL

Scandinavian Developer Conference (Java Track)
APRIL 16–19, GOTHENBURG, SWEDEN
SDC2012 brings together people who work in software development to exchange experiences and discuss new ideas and views on industry trends. A dedicated Java track will cover the latest Java tools and frameworks. Tutorials are new to this year's agenda; topics include agile development, innovation games, and more.

JavaOne Russia
APRIL 17–18, MOSCOW, RUSSIA
JavaOne Russia is a regional event that allows learning about all aspects of Java—from better programming with the new features of Java SE 7 to using other languages on the JVM. Learn from the experts about the Java roadmap, how to use Java more effectively, and how to choose and use tools in the Java ecosystem for your development work. Meet with other Java developers and explore new possibilities.

Devoxx France
APRIL 18–20, PARIS, FRANCE
Devoxx France is a developer conference organized by the Paris Java user group (JUG) team. A quarter of the sessions will be in English and the rest in French. Tracks include Java, Java EE, and architecture; Web, mobile, and the cloud; alternate languages on the JVM; and companies, software development methods, and techniques.

MAY

JavaOne India
MAY 3–4, HYDERABAD, INDIA
JavaOne India is a regional event that allows learning about all aspects of Java—from better programming with the new features of Java SE 7 to using other languages on the JVM. Learn from the experts about the Java roadmap, how to use Java more effectively, and how to choose and use tools in the Java ecosystem for your development work. Meet with other Java developers and explore new possibilities.

Adopt-a-JSR in Full Swing


Ben Evans

Martijn Verburg

Java user groups (JUGs) around the world are signing up for the Adopt-a-JSR initiative, engaging their local developer communities to grow Java through active participation in the Java Specification Requests (JSRs) of the Java Community Process (JCP). Most recently, the Houston Java Users Group adopted JSR 321 (Trusted Computing API for Java), and Java User Group Chennai adopted JSR 331 (Constraint Programming API).

Ben Evans and **Martijn Verburg** of the London Java Community have been on the road at conferences and local JUG meetings introducing the Adopt-a-JSR initiative. The objective of the program is to encourage JUG members to get involved in a JSR and to evangelize that JSR to their JUG and the wider Java community in order to increase grassroots participation.

To get started, visit the Adopt-a-JSR program on Java.net. See the matrix of JSRs and the JUGs that are participating in those JSRs to get an idea of which JSRs might benefit most from adoption by your JUG.

BEN EVANS' PHOTOGRAPH BY CATHERINE CURRIE

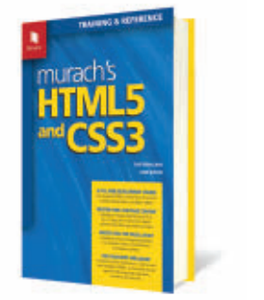
JAVA BOOKS

JAVA 7 RECIPES
By Josh Juneau, Mark Beaty, Carl Dea, Freddy Guime, and John O'Conner
Apress (January 2012)

Java 7 Recipes offers solutions to common programming problems encountered every day while developing Java-based applications. Fully updated with the newest features and techniques available, *Java 7 Recipes* provides code examples involving Servlets, JavaFX 2.0, XML, Java Swing, and much more. Content is presented in the popular problem/solution format. Look up the programming problem that you want to solve, read the solution, and apply the solution directly in your own code. Problem solved!

PRO JAVAFX 2 PLATFORM
By James Weaver, Weiqi Gao, Stephen Chin, Dean Iverson, and Johan Vos
Apress (March 2012)

In *Pro JavaFX 2 Platform: A Definitive Guide to Script, Desktop and Mobile RIA with Java Technology*, the authors show how you can use the JavaFX platform to create rich client Java applications. Covering the JavaFX API, development tools, and best practices, this book provides code examples that explore the exciting new features provided with JavaFX 2.0. It contains engaging tutorials that cover virtually every facet of JavaFX development, and reference materials on JavaFX that augment the JavaFX API documentation. *Pro JavaFX 2 Platform* is an essential guide to JavaFX 2.0.



MURACH'S HTML5 AND CSS3
By Zak Ruvalcaba and Anne Boehm
Mike Murach & Associates (December 2011)

HTML5 and CSS3 offer dramatic improvements for providing the Web experience that users want today—improvements that no Web developer should miss. If you're a beginner creating your first Website, this book will teach you the basics of how to accomplish it in just the first six chapters. If you're an experienced Web designer, JavaScript programmer, or server-side programmer, you won't find a better way to upgrade your skills and your sites than with this book. Read a sample chapter.

JAVA EE6 COOKBOOK FOR SECURING, TUNING, AND EXTENDING ENTERPRISE APPLICATIONS
By Mick Knutson
Packt (February 2012)

This book provides recipes for securing, tuning, and extending enterprise applications using a Java EE 6 implementation. The author begins with the essential changes in Java EE 6, dives into the implementation of some of the new features of the Java Persistence API (JPA) 2.0 specification, and covers how to implement auditing for relational data stores. Several additional sections describe some of the subtle issues encountered, tips, and extension points for stating your own JPA application or extending an existing application.

Data Quality Tools for Java

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

The New Java Developers

Meet some of the young developers creating the future of Java. **BY TORI WIELDT**

As the Java developer community lead for Oracle, the best part of my job is going to Java conferences and getting to know developers. I've had the pleasure of meeting developers who are smart, fun, and passionate about Java—these individuals make the Java community happen. Here we introduce you to some of the youngest Java developers I've met around the world. They range in age from 18 to 27 and have been drawn to this field by Java's flexibility, breadth, and energetic open source community. It's great to see a whole new generation of Java developers contributing to the vibrancy of our community. The future is in good hands.

—Tori Wieldt, Java Developer Community Lead, Oracle

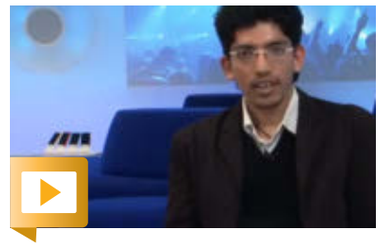
ON-LOCATION PHOTOGRAPHY: DIEGO VERGÉS REQUEJO/GETTY IMAGES, EAST AND WEST JAVA, INDONESIA; NAMAS BHOJANI, BANGALORE, INDIA; PIOTR MALECKI/GETTY IMAGES, KRAKÓW, POLAND; PAULO FRIDMAN, SÃO PAULO, BRAZIL; OLEG NIKISHIN/GETTY IMAGES, VLADIMIR, RUSSIA; JAMES OATWAY/GETTY IMAGES, JOHANNESBURG, SOUTH AFRICA; BOB ADLER, SAN FRANCISCO LOCATION

01 RAM KASHYAP

Bangalore, India–based developer Ram Kashyap loves creating applications for his phone. "If I can't do something in the real world, I can create it in the digital world, and that inspires me a lot," Kashyap says.

For the past 10 months, Kashyap has been working diligently to get educational tools onto mobile phones in an effort to increase the global literacy rate for children ages 4 to 18. He targets phones that are Java app–enabled, because these devices can be found in huge numbers in developing countries.

Kashyap is founder and head of the Student Nokia Developer group, a student community that develops high-quality mobile applications for Nokia phones. He's trained nearly 400 students all over India in developing on Nokia-based technologies. At JavaOne San Francisco 2011, Kashyap presented some of the applications the group has created in two birds-of-a-feather sessions. You can visit Student Nokia Developer to see sample applications (don't miss RPS Lizard Spock, a fun take on the classic game rock-paper-scissors) and learn more about the group on its Facebook page. The Student Nokia Developer group is now full of passionate student developers from all over the world, thanks to Kashyap's efforts at inspiring and motivating these young people and showing them the resources they can use to gain design, analysis, development, testing, management, and marketing skills. As if programming and his work with the group didn't keep him busy enough, Kashyap also sings classical Indian music, plays cricket, and goes on long bike rides.



Meet Ram Kashyap.

PHOTOGRAPH BY NAMAS BHOJANI

SNAPSHOT

Age 21 | **Occupation** Student at the People's Education Society Institute of Technology | **Location** Bangalore, India

02 GALINA PROSKURINA

SNAPSHOT

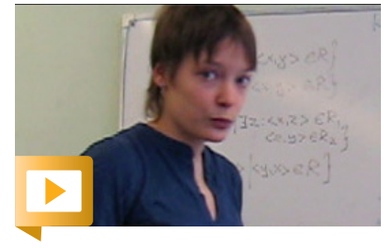
Age 27 | **Occupation** Teaching assistant at Vladimir State University
Location Vladimir, Russia

Galina Proskurina wrote her first program in Basic in 2001, at the age of 16; it was a psychological test to determine if the user has a normal relationship with his or her mother, replicated from a multiple-choice quiz in a women's magazine. "My classmates were impressed by the visual effects—in the opening, I had the letters in the name of the test dance onscreen," Proskurina says.

Since then, Proskurina has written many applications, including the prototype of a decision-support system for a gas transport company. Currently she's a teaching assistant at Vladimir State University in Vladimir, Russia (180 km from Moscow), where she teaches Java, SQL and MySQL, and mathematics. In addition to her work with university students, Proskurina teaches programming to high school students—she's currently instructing them on how to build simple utilities for mobile phones using Java ME.

The most important concept about programming that Proskurina learned in school, she says, was to use libraries. She initially wrote all her programs from scratch, but learning Java EE gave her another view of programming: as the "assembler" of existing parts. "The most important part of Java programming is to understand what is already built and what you have to build yourself," Proskurina explains.

When not teaching or coding, Proskurina loves being active. She likes to ski, swim, and ride her bike, all of which "make me feel like I'm flying," she says. In the future, she would like to be an in-demand, "cool" programmer and work on a project with significant societal impact. Someday, she hopes, one of her programs might even save the world.



Meet Galina Proskurina.

PHOTOGRAPH BY OLEG NIKISHIN/GETTY IMAGES

SNAPSHOT

Nety Herawaty (top) | **Age** 18 | **Occupation** Student at Gunadarma University | **Location** Subang, West Java, Indonesia

Mila Yuliani | **Age** 21 | **Occupation** Developer at Meruvian Foundation | **Location** Malang, East Java, Indonesia

03 NETY HERAWATY
04 MILA YULIANI

At age 15, Nety Herawaty was chosen from a group of 500 IT students to attend Oracle Academy classes in programming, database design, and business at her high school.

Herawaty is now a freshman at Gunadarma University in Jakarta, Indonesia. She believes object-oriented programming is the basis of Java, and now that she understands the basics, she says, it will be easier for her to learn advanced Java.

Herawaty sees women as having an essential role to play in the future of IT. Driven by this belief, she helped found Java Duchess Indonesia (JDuchess Indonesia) with Mila Yuliani in November 2011. JDuchess Indonesia is a network for connecting Indonesian women in Java technology with each other and with the global Duchess community. Herawaty would like more Indonesian women who are interested in Java technology to join them, as she is eager to share her knowledge and experience with others.

When coding, Herawaty likes listening to music by Avril Lavigne, Simple Plan, and Secondhand Serenade.

Mila Yuliani, cofounder of JDuchess Indonesia, loves programming. At Gunadarma University, she learned Java, PHP, Visual Basic, C++, and SQL. She now works as a Java programmer and trainer at Meruvian Foundation, a nonprofit dedicated to teaching technology in Indonesia. As a programmer, Yuliani wants to specialize in Java EE and hopes to become a CTO someday.

Yuliani likes working on new technology challenges, such as how to create an application in Ruby or Python. She loves when her program throws an error, so she can use debug and trace to figure it out, and she's also interested in RFID and smartcards.

Full of enthusiasm, when Yuliani exclaims, "Nothing is impossible when you try!" you believe her.

Meet Mila Yuliani.

PHOTOGRAPH BY DIEGO VERGÉS REQUEJO/GETTY IMAGES

SNAPSHOT

Age 25 | **Occupation** Software engineer at Citibank
Location São Paulo, Brazil

05 LOIANE GRONER

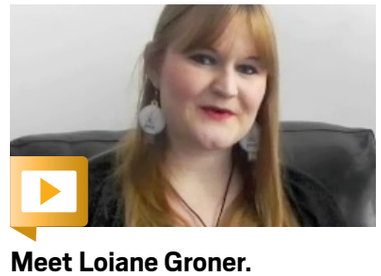
When Loiane Groner was 10 years old, she got her first computer and fell in love with it. Spurred on by that first love, she signed up for a computer science class in college, only to discover that she had to do something called *programming*. "I discovered that I could 'talk' to a computer and make it do things I would like it to do," she says, "and it was amazing!" Groner is now a published technical author and Java user group leader in Brazil.

Groner currently works at Citibank in São Paulo developing enterprise applications. "The fact that Oracle is moving Java forward is great news for me," she says, adding that she likes Java for its ease of integration with other technologies, no matter which Java framework or API she's working with. Groner is tracking Oracle's Project Avatar, which will bind HTML5 with Java SE, Java EE, and Java ME, making it possible for developers to build hybrid applications. She's also interested in learning more about robotics, artificial intelligence, and heuristic algorithms.

Groner also likes JavaScript, and she loves to work on projects that integrate both Java and JavaScript. This interest led to the publication of her first book: *Ext JS 4 First Look* (Packt Publishing, 2011), a guide to the new features in the JavaScript framework Ext JS 4. She's currently writing a second book for Java students.

Groner loves heavy metal, and when she's programming, some of the bands she likes to listen to are Iron Maiden, Metallica, Blind Guardian, and Nightwish. She has two blogs (the English-language Loiane Groner: My Development Notes and, in Portuguese, Loiane Groner: Java, Ext JS, desenvolvimento e tecnologia) where she shares her experiences with new technologies.

When she's not writing code, blog posts, or books, she loves to get out in nature and go hiking. Not one to stay idle, she's also learning photography.



Meet Loiane Groner.

PHOTOGRAPHY BY PAULO FRIDMAN

SNAPSHOT

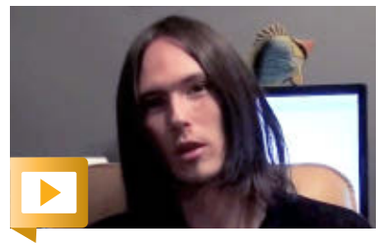
Age 23 | **Occupation** Developer at SoftwareMill | **Location** Kraków, Poland

06 KONRAD MALAWSKI

Konrad Malawski likes to keep learning and pushing his own limits. He admits being "weirdly attracted" to beautiful code and type theory and was known at his university as "the Java guy." What's behind the attraction? For one thing, Malawski likes that there are many stable and open source libraries available for Java. "I once found myself debugging deep into the standard networking libraries. Having the sources available made finding the overly weird bug a breeze. It would have been a pain without source code access," he explains.

Malawski wants to continue strengthening the Java community in Poland, where he's an active member of the Polish Java User Group and a speaker and organizer of programming-related events. He's also part of the planning committee for GeeCON, a conference focused on Java and Java Virtual Machine–based technologies that is held in Poland every May. In the future, Malawski looks forward to learning more about Java so he can both be a better developer and share his knowledge with other developers at workshops, code retreats, and conferences. You can read about the many projects he's involved in on his blog, Blog.Project 13.

Malawski usually listens to music when he codes, streaming JRock (Japanese rock) and New Age or ambient music. As for hobbies, it's quite rare for Malawski to not be hacking or working with the Java community. Programming is not just his job; it's what he loves to do. "I got lucky with my hobbies and work," Malawski says. "They're both the same thing."



Meet Konrad Malawski.

PHOTOGRAPH BY PIOTR MALECKI/GETTY IMAGES

SNAPSHOT

Age 23 | **Occupation** Software developer at Barone, Budge & Dominick
Location Johannesburg, South Africa

07 HEIN SMITH

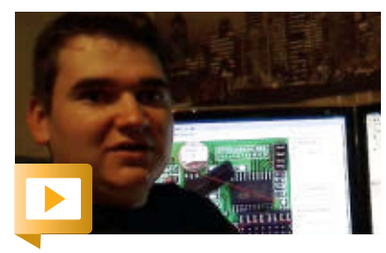
Hein Smith tackled his first development project at age six, when he built a functioning cable car out of LEGO blocks and sent it from the second floor of his house to a tree in the backyard. He loves challenges, whether it's creating a design, bug-fixing using Java EE APIs, or lock-picking ("for educational purposes only!" he insists).

Smith liked Java early on because of its flexibility. While working on a personal project testing how one could break weak encryption mechanisms, he needed to merge various word lists (with the support of different file-encoding input) on the Windows platform and could not find a working application that suited his needs.

He ended up using Java, because it made it so easy to get the job done. More recently, his projects have included creating HS File Hasher, a multiplatform SHA and MD5 checksum utility, and Kommunika, a Java API for two-way secure communication with an Arduino device. In the future, Smith plans to create a Web-based home automation system, both to keep up with the latest Java technologies and to play with the Arduino platform.

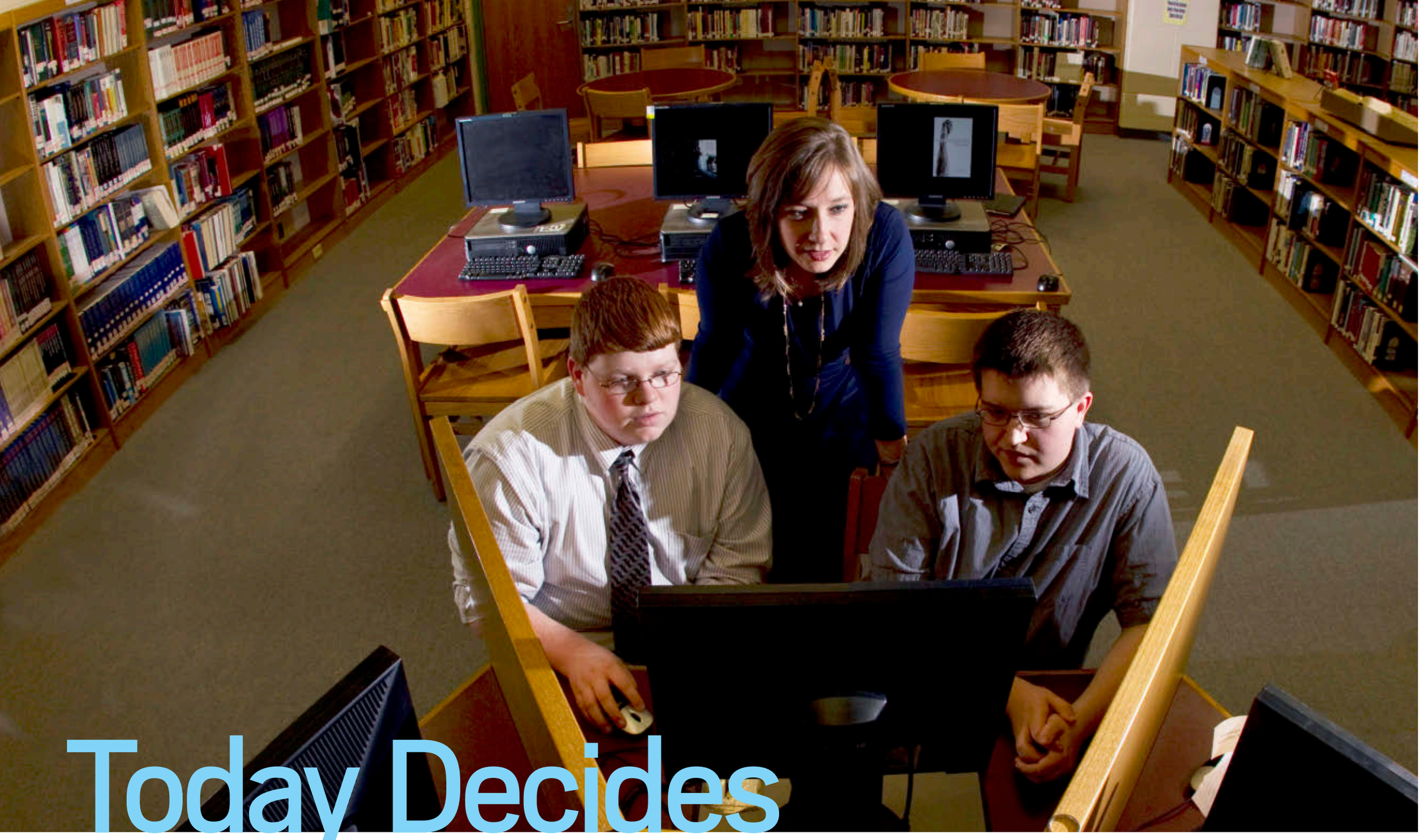
Smith obtained his certification in Java SE 6 programming in December 2010—his first certification from Oracle and "hopefully one of many to come," he says. Next he wants to get his Oracle Certified Expert, Java EE 6 JSPs and Servlet Developer certification. Certification alone isn't enough, Smith says, noting the importance of seeing the power of Java implemented in real-world scenarios. Smith is also interested in enterprise security and would one day like to become a specialist in security for Java enterprise applications.

When coding, Smith gets sustenance from "regular cups of good, freshly ground coffee." And when he's not solving problems, he loves to go mountain biking.



Meet Hein Smith.

PHOTOGRAPH BY JAMES OATWAY/GETTY IMAGES



Today Decides Tomorrow

The Oracle Academy helps students embrace Java.

BY RICH SCHWERIN

PHOTOGRAPHY BY MARK CORNELISON/GETTY IMAGES

At Woodford County Public Schools, technology is key to increasing student engagement.

For some 4,000 students across seven campuses in the heart of Kentucky bluegrass country, few barriers remain between learning and technology. That's due in large part to Woodford County Public Schools' focus on the use of technology to increase student engagement, individualize learning, provide immediate feedback, and deliver 24/7 access to resources.

"Our school recently introduced an initiative where each student has his or her own Apple iPad, which enriches the whole classroom experi-

Students Chris (below left) and Caleb learned to develop using Java with the support of Becky Keith, technology resource teacher.

BY THE NUMBERS: THE ORACLE ACADEMY

US$2 billion
The value in software, curriculum, hosting services, and teacher professional development that the Oracle Academy delivers each year.

95 The number of countries the Oracle Academy serves worldwide.

1.5 million The number of students the Oracle Academy reaches worldwide.

ence," says Caleb, a Woodford County High School student and member of the Student Technology Leadership Program (STLP). "There's also a Moodle online learning management system, a state-of-the-art media program with video cameras and editing software for broadcast and yearbook, plus a wide range of apps and Websites to organize and manage notes, information, projects, and more."

As STLP members, Caleb and fellow student Chris not only take the lead in technology use at their school, but also use their skills to enter competitions. They were recently awarded second place in the "16 and under" age group of an application development competition sponsored by the Oracle Education Foundation. Their winning entry, "Helping Hand: Find a Worthy Cause," is a Java-based application that helps nonprofit organizations by connecting potential volunteers and donors to those organizations.

"We wanted to spread awareness that organizations such as the Humane Society are just as dependent on volunteers as they are on monetary contributions. And we learned that small nonprofits spend significant resources trying to reach potential volunteers and donors, so we wanted to address these challenges," says Chris. "Our Web app aggregates nonprofit information from a variety of sources into a profile page that is easy for supporters to find, through categorization and keywords."

JAVA: VERSATILITY AND POWER

Initially, Caleb and Chris considered using PHP for their project, but they instead chose Java in order to take advantage of the many Java libraries, frameworks, integrated development environments, and application servers available, and because of Java's versatility and power.

"I use Java for a lot of personal projects but had never used it for developing a Web application. Since I was already familiar with the language, it wasn't difficult to learn the necessary skills," says Chris. "And during the four months we spent working on our entry to the competition, we also learned to use other tools, including Mercurial SCM, Google Documents List API, Oracle VM VirtualBox, Apache Tomcat, and Oracle GlassFish Server."

For Becky Keith, Woodford County Public Schools' technology resource teacher, projects like this address both the school district's goals and the students' needs.

"Not only did the students further develop their technology skills, but they were also challenged to solve a problem in the commu-

Clare Dolan chats with developer Lai Heng Chua about the Oracle Academy Java curriculum.

EXAM CREDIBLE

"Our Java curriculum maps to both the Advanced Placement [AP] Computer Science A exam and Oracle's Java certification exam." *—Clare Dolan, vice president of corporate citizenship at Oracle*

PHOTOGRAPH BY BOB ADLER

nity," says Keith, who coached the students. "Chris and Caleb spent a lot of time researching and interviewing nonprofit organizations to determine their needs, and they were able to use technology to promote the needs of nonprofits. I had two intrinsically motivated and talented students, so it is rewarding to see their hard work recognized."

THE ORACLE ACADEMY

Facilitating this sort of rewarding educational experience while preparing students for future careers are just two facets of the Oracle Academy, which offers education institutions a complete portfolio of software, curriculum, hosted technology, faculty training, support, and certification resources.

"Oracle works to advance education with state-of-the-art technology programs, and our primary vehicle for this is the Oracle Academy," says Clare Dolan, vice president of corporate citizenship at Oracle. "Additionally, Oracle also makes available select cash grants to organizations that support computer science and engineering education."

Dolan explains that the Oracle Academy commitment equates to about US$2 billion annually, through in-kind contributions of technology, curriculum, and services. The Oracle Academy, launched in 2001, now reaches more than 1.5 million students in 95 countries worldwide.

"The Oracle Academy offers high schools, community colleges, and four-year colleges and universities the ability to integrate Oracle technology and curriculum into their computer science, engineering, and business curricula," says Dolan. "It is a complete offering, including teacher training, opportunities for schools and teachers to showcase their excellence, and community and mentoring opportunities."

The Oracle Academy delivers three programs: Introduction to Computer Science, which is designed for high schools and community colleges and includes a structured curriculum to help students master entry-level technical skills; Advanced Computer Science, which is designed for university computer science departments and provides students hands-on access to Oracle database and middleware software; and Enterprise Business Applications, which is designed for university computer science departments and business schools and gives students

Teens in Tech Labs

Since 2008, Teens in Tech Labs has been providing tools and resources for young entrepreneurs worldwide to encourage innovation. Created by 18-year-old entrepreneur Daniel Brusilovsky, the Mountain View, California–based company was founded on the premise that entrepreneurship doesn't happen at a certain age or in a certain location; it happens at all ages, everywhere. Teens in Tech Labs offers a resource center, filling the gap before college when resources are slim for young entrepreneurs, and is funded by corporate sponsors to provide customized programs, onsite conferences, and additional outreach. Teens in Tech Labs also manages three other initiatives: the annual Teens in Tech Conference, which brings together youth and technology in one place for a day focused on connecting youth and technology with entrepreneurship, startups, and the Web; the Teens in Tech Incubator, a summer incubator program helping five teams of young entrepreneurs launch five products over the course of a summer; and Teens in Tech Connect, a program focused on connecting young entrepreneurs.

Java-Based ROBOTICS COMPETITIONS

For Inspiration and Recognition of Science and Technology (FIRST) is a nonprofit organization that helps young people discover and develop a passion for science, engineering, technology, and math. Founded more than 20 years ago by inventor Dean Kamen, the 2009–2010 FIRST season attracted more than 210,000 youth and more than 90,000 mentors, coaches, and volunteers from 56 countries. The annual programs culminate in an international robotics competition and celebration, where teams win recognition, gain self-confidence, develop people and life skills, make new friends, and perhaps discover an unforeseen career path.

The Java SDK for FIRST Robotics is based on the Squawk Java Virtual Machine (JVM). Inspired by the Smalltalk Squeak project, the main goal of the Squawk virtual machine project is to write as much of the virtual machine as possible in Java, for portability, ease of debugging, and maintainability. Traditionally, most JVMs are written in C/C++. Squawk aims at pushing the bar and writing most of the JVM in Java.

Greenlight for Girls is an international nongovernmental organization and a social initiative with the mission to encourage young girls of all ages to consider a future in math, science, engineering, and/or technology by introducing them to the world of science in fun and exciting ways. At Greenlight@Brussels Day 2011, Tasha Carl and other members of Brussels Java User Group—supported by the Paris, France–based European Space Agency and Genoa, Italy–based Scuola di Robotica (School of Robotics)—organized a successful workshop as part of EU Robotics Week. Teams of girls in Brussels, Belgium, and Genoa programmed robots for a simulated "Mission on Mars" as part of an event reaching 300 girls, inspiring them to study science and technology.

hands-on access to Oracle applications that are widely used in a variety of industries.

The Oracle Academy's Introduction program delivers semester-based curriculum that can be easily incorporated into secondary school computer science programs. Later this year, the Oracle Academy is releasing a new Java curriculum for secondary school students.

"Few subjects will open as many doors for students in the twenty-first century as computer science and engineering," says Dolan. "As the steward of Java, Oracle is uniquely positioned to help educators awaken and deepen students' interest in these subjects."

NEW JAVA CURRICULUM, PLUS ALICE AND GREENFOOT

The Oracle Academy's Java curriculum targets students with little or no previous programming experience. The two-semester curriculum starts with Java Fundamentals and is followed by Java Programming.

"Our Java curriculum maps to both the Advanced Placement [AP] Computer Science A exam [delivered in the U.S.] and Oracle's Java certification exam," explains Dolan.

As a complement to the Java curriculum, the Oracle Academy looks for innovative, engaging ways to promote Java and technology education. Oracle supports Alice and Greenfoot, two interactive Java programming environments for students. "Alice and Greenfoot are designed to be fun and engaging—they really grab students' attention. When young people use these applications, they are immediately successful and, in turn, become highly motivated to learn more," says Dolan.

For students ages 10 through 22, Alice introduces object-oriented programming through a drag-and-drop programming environment that allows students to create animation for telling a story, playing an interactive game, or sharing a video on the Web. For students at the secondary and undergraduate levels, Greenfoot provides a Java development environment that enables easy creation of two-dimensional graphical applications, such as simulations and interactive games.

CHOOSING JAVA

"I use Java for a lot of personal projects but had never used it for developing a Web application. Since I was already familiar with the language, it wasn't difficult to learn the necessary skills."
—Chris, student, Woodford County High School

While the new Oracle Academy Java curriculum and initiatives like Alice and Greenfoot foster early student interest in computer science and Java learning opportunities, the Oracle Academy also works to ensure that college/university students gain industry-relevant skills prior to entering the workforce.

"Through our Advanced Computer Science program, we provide four-year colleges and universities software and curriculum across Oracle's technology stack," says Dolan. "We intend to support colleges and universities to incorporate Java as a standard component within their computer science curricula."

And because Java is the foundation for virtually every type of networked application and is the global standard for developing and delivering mobile and Web apps, games, and enterprise software, students proficient in Java will have even more opportunities. `</article>`

Rich Schwerin is a senior manager with Oracle Publishing who focuses on social media.

JavaMail Delivers

Use JavaMail and Facelets to create a Web application that can send e-mail.

T. LAMINE BA

BIO

In this article, we are going to build a simple Web application that uses the core JavaMail API to send e-mail. The application includes three Web pages: a front page, a "sent e-mail" confirmation page, and a "failed e-mail" notification page.

Note: The source code for the application can be downloaded here.

The front page of the Web application (see **Figure 1**) contains the following Web components:

- An input field for the e-mail address of the sender
- An input field for the e-mail address of the recipient
- An input field for the subject of the e-mail
- An input field for the body of the e-mail
- An input field for the IP address or host name of the Simple Mail Transfer Protocol (SMTP) server
- An input field for the username required by the SMTP server
- An input field for the password required by the SMTP server
- An input field for the port number used by the SMTP server
- A button to send the e-mail

The confirmation page (see **Figure 2**) and the notification page (see **Figure 3**) need only a button that redirects the user back to the front page.

MORE THAN TEXT
The JavaMail API uses the **JavaBeans Activation Framework** to handle non-plain-text content.

About the JavaMail API

The JavaMail API is a package that provides general e-mail facilities, such as reading, composing, and sending electronic messages. JavaMail, which is a platform-independent and protocol-independent framework, is included in Java EE.

As shown in **Figure 4**, JavaMail has an application-level interface used by the application components to send and receive e-mail. There is also a service provider (SP) interface that speaks protocol-specific languages. For instance, SMTP is used to send e-mails. Post Office Protocol 3 (POP3) is the standard for receiving e-mails. Internet Mail Access Protocol (IMAP) is an alternative to POP3.

In addition, the JavaMail API contains the JavaBeans Activation Framework (JAF) to handle e-mail content that is not plain text, including Multipurpose Internet Mail Extension (MIME), Uniform Resource Locators (URLs), and file attachments.

Figure 1

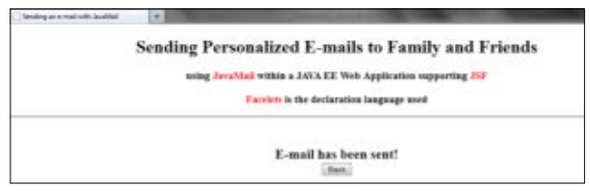


Figure 2

Figure 3

Required Software

For the purposes of this tutorial, the following software is used:

- The Microsoft Windows platform (I used Microsoft Windows 7.)
- The JDK for Java EE 6 or higher
- A common IDE such as the NetBeans IDE (I used version 7.)
- A Web server such as Oracle GlassFish Server, GlassFish Server Open Source Edition, or Apache Tomcat

Methodology

The tutorial uses JavaServer Faces (JSF) technology to build the Web

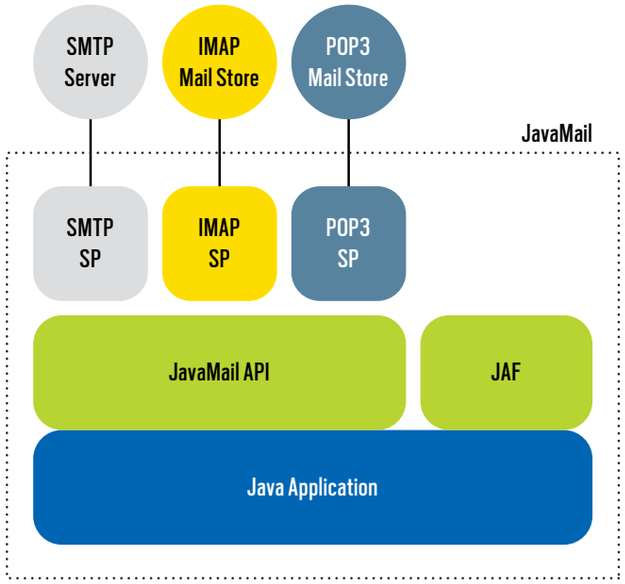


Figure 4

application. Accordingly, the following workflow is proposed:

- Step 1: Create a backing bean.
- Step 2: Create Web pages using component tags.
- Step 3: Map the FacesServlet instance.

Creating a Backing Bean

A backing bean is a type of managed bean specific to the JSF technology. It holds the logic of the Web application and interacts with the Web components contained in the Web pages. The backing bean can contain private attributes that correspond to each Web component, getter and setter methods referring to the attributes, and methods to handle the following four tasks:

- Perform processing associated with navigation from one Web page to another
- Handle action events
- Perform validation on a component's value
- Handle value-change events

Accordingly, in a backing bean called emailJSFManagedBean, we create the getter and setter methods necessary for each of the Web components listed earlier (except the Send E-mail button). If the recipient's e-mail address is a variable of type String called to, **Listing 1** shows how the getter and setter methods would be defined.

@ManagedBean, as shown in **Listing 1**, is a declaration that registers the backing bean as a resource with the JSF implementation. In addition, @RequestScoped is the annotation that identifies the managed bean as a resource that exists only in the scope of the request. In other words, the bean exists for the duration of a single HTTP request for the user's interaction with the Web application.

We also create two specific methods within the backing bean.

The first method's function is to validate all e-mails submitted by the user on the Web application (see **Listing 2**).

```
message = "E-mail is required";
```

Note that the validation e-mail method takes three arguments:

- The context of the JSF implementation, in order to pass error messages from the managed bean to the user interface
- The identifier UIComponenttoValidate of the Web component that is invoking the method, which, in this case, is a text input field (see **Listing 1**) method that takes user input as an argument—user input is illustrated in **Figure 1**
- The variable value, which contains the e-mail address that needs to be validated

Accordingly, the code shown in **Listing 2** accomplishes the following tasks:

- It gets the local value of the Web component.
- It checks whether the value is null or empty.
- If the value is null or empty, the method sets the component's valid property to false and it sets the error message to E-mail address is required.
- Otherwise, the method checks whether the @ character and the

E-MAIL EASE
The JavaMail API is a package that provides general e-mail facilities, such as **reading, composing, and sending** electronic messages.

LISTING 1 **LISTING 2**

```
package useJavaMail;
/*** Import all necessary libraries ***/

@ManagedBean
@RequestScoped
public class emailJSFManagedBean {
    private String to;
/** Creates a new instance of emailJSFManagedBean */
  public emailJSFManagedBean() {
    to = null;
  }

  /**
   * @return the to
   */
  public String getTo() {
    return to;
  }
  /**
   * @param is the "to" to set
   */
  public void setTo(String to) {
    this.to = to;
  }
}
```

Download all listings in this issue as text

period (.) character are contained in the value.

- If they aren't, the method sets the component's valid property to false and it sets the error message to E-mail address is invalid.

Then, the error message is sent to the FacesContext instance, which associates it to the invoking Web component.

The second method handles the logic for sending an e-mail with JavaMail. This navigation handling method is triggered by the action of clicking the Send E-mail button (see **Listing 3**).

This is known as an action method. It is a public method that takes no argument and returns a string that corresponds to the page that the Web application will navigate to. In this case, the method produces and sends an e-mail. If the e-mail transmission is successful, the method returns g_response (for "good response"), which displays the page g_response.xhtml in the browser (see **Figure 2**). If the e-mail transmission fails, b_response (for "bad response") is returned and the browser displays the page b_response.xhtml (see **Figure 3**).

INDEPENDENCE DAY
JavaMail is **platform** and **protocol** independent.

In order to send an e-mail using JavaMail, we first initiate an e-mail session instance with the Session class. The e-mail session is the starting point for JavaMail. It uses the java.util.Properties class to get information, such as the e-mail server, the username, and the password, which can be shared across the rest of the application. In this case, we create a default instance of the Session class:

```
session =
Session.getDefaultInstance(props, null);
```

Second, through the session, we produce the e-mail using the Message class. However, considering that Message is an abstract class, we choose instead its subclass MimeMessage, which allows us to create messages that understand MIME types and headers, as defined in the different Requests For Comments. The message is constructed with the session as an argument:

```
MimeMessage message =
new MimeMessage(session)
```

Then, we send the e-mail by manipulating an object of type Transport. The message is sent via the transport protocol SMTP. The transmission is handled by the Transport class, and an object is instantiated as follows:

```
Transport transport =
session.getTransport("smtp");
```

Then, the transport object attempts to connect to the SMTP server using the suggested credentials (the SMTP server address, the port number that accepts SMTP connections, the username, and the password) to pass authentication on the server.

LISTING 3

```
public String submitEmail() {
    // create e-mail and send
    /*** Initialize variables ***/
    props = new Properties();
    // fill props with session and message information
    session = Session.getDefaultInstance(props, null);
    message = new MimeMessage(session);
    try {
      message.setContent(this.getDescr(), "text/plain");
      message.setSubject(this.getSubject());
      fromAddress = new InternetAddress(this.getFrom());
      message.setFrom(fromAddress);
      toAddress = new InternetAddress(this.getTo());
      message.setRecipient(RecipientType.TO, toAddress);

      // Transport message
      message.saveChanges(); // implicit with send()
      Transport transport = session.getTransport("smtp");
      transport.connect(this.smtp, this.port, this.username, this.password);
      if(transport.isConnected() == false)
        return "b_response";
      transport.sendMessage(message, message.getAllRecipients());
      transport.close();

    }
    catch (MessagingException me) {
      // handle catch
      return "b_response";
    }

    return "g_response";
  }
```

Download all listings in this issue as text

```
transport.connect(this.smtp,
this.port, this.username,
this.password);
```

If the SMTP server accepts the connection, the e-mail is sent via send.

Finally, we close the transportation service by invoking the close command:

```
transport. sendMessage(message,
message.getAllRecipients());
transport.close();
```

Note: The file containing the backing bean should be under the Sources Packages directory of the Web application.

Creating Web Pages

The different Web pages of the application take advantage of the Facelets declaration language to produce tags for various Web components.

Create the front page. On this page, there are four types of tags associated to the Web components: inputText, inputSecret, inputTextArea, and commandButton.

The inputText is equivalent to an input tag of type text in HTML. We use this type of tag to obtain the sender's address, the recipient's address, the subject of the e-mail, the SMTP server address, the SMTP server username, and the port number of the SMTP server.

The inputSecret is equivalent to the input tag of type password in HTML. It is also a field that takes user input. However, in contrast to the inputText tag, the inputSecret does not display the value entered by the user. This tag is used to record the SMTP server password.

The inputTextArea is equivalent to the textarea tag in HTML. It is used to record the body of the e-mail we intend to send.

User input is validated by the application either by using the standard validators or by invoking a validating method implemented in the backing bean (refer to **Listing 2**).

For example, using Facelets, we invoke the validating method emailJSFManagedBean.validateEmail for the FROM address field using the code shown in **Listing 4**.

Note that the purpose of the message tag (<h:message/>) is to display the error message when the e-mail address validation fails (see **Figure 5**).

As another example, **Listing 5** shows how we use standard validators in Facelets for the SUBJECT field.

The validateRequired tag (<f:validateRequired/>) is applied to the inputText with an id of subject. This invalidates the form when it is submitted and the SUBJECT field is empty. In this case, an error message is displayed where the message tag (<h:message/>) is located (see **Figure 6**).

Figure 5

Figure 6

Create the confirmation page and the error notification page. The confirmation page (g_response.xhtml) is called when an e-mail has been sent (see **Figure 2**). On the other hand, the error notification page (b_response.xhtml) is called when the e-mail transmission fails (see **Figure 3**). Both pages contain only one Web component, which is a Facelets commandButton tag:

```
<h:form>
 <h:commandButton id="back"
value="Back" action="index">
</h:form>
```

The code creates a button that, when clicked, forwards the user to the front page (index.xhtml).

Mapping the FacesServlet Instance

The final step consists of mapping the FacesServlet instance by altering the Web deployment descriptor, that is, the web.xml file. **Listing 6** is a typical example.

Note, however, that the mapping is done automatically if you are using an IDE such as NetBeans.

LISTING 4 LISTING 5 / LISTING 6

```
<h:form>
  <table>
    <tr>
      <th style="width:100px" align="right">FROM:</th>
        <td>
          <h:inputText id="from" size="100"
validator="#{emailJSFManagedBean.validateEmail}"
value="#{emailJSFManagedBean.from}" />
          <span style="margin-left:10px"><h:message style="color:red"
for="from"/></span>
        </td>
      </tr>
  </table>
</form>
```

Download all listings in this issue as text

Conclusion

In this article, you learned how to build a simple Web application that uses the core JavaMail API to send e-mail. Now you can build more-interactive Web applications where e-mailing can be customized and integrated in various forms. </article>

LEARN MORE

- NetBeans online help
- *The Java EE 6 Tutorial: Basic Concepts*, fourth edition (Prentice Hall, 2010)
- "Overview of JavaMail APIs"

MICHAEL KÖLLING

BIO

The Snakes Are After Us!

Improving our turtle game by making the snakes a little more agile

When we left our Java project in the previous issue of *Java Magazine*, we had a little, playable game in which a turtle was eating pizza while trying not to get eaten by the snakes (see **Figure 1**).

If you haven't been coding along with our project so far, download and install Greenfoot, and then download our project (as it is so far) and join in this session.

Improving the Snakes

In our current version, the snakes' movement is controlled by these two lines of code:

```
move(4);
turn(Greenfoot.getRandomNum-
ber(40)
-20);
```

The effect is that snakes always move forward, and they turn at a random angle (up to 20 degrees left or right) at every step.

This works pretty well when a snake is in the middle of the screen. However, when it hits the edge of our world, it gets stuck there for a while before managing to turn around and chase the turtle again. We can improve this by making snakes always turn around when they reach the edge of the world.

(This also gives me a chance to show you two things: We can use an if statement to get a bit more practice with these, and we get an opportunity to call methods from the world object.)

Turning at the Edge of the World

If we want to make the snake turn at the edge of the world, we first have to recognize that we are at the edge. We can do this by defining a new method called atWorldEdge that checks this. This method returns a Boolean result: true if we are near the edge, and false if we are not. The signature looks like this:

```
public boolean atWorldEdge()
```

Within this method, we want to write some code that expresses something like this:

- If we are close to the left edge or the right edge, return true.
- If we are close to the top or the bottom, return true.
- Otherwise, return false.

For the first test mentioned here, we need to know our *x* coordinate. For the second test, we need our *y* coordinate. We can get both of these using the getX() and getY() methods of the Actor class. These two methods return as their result our current position.

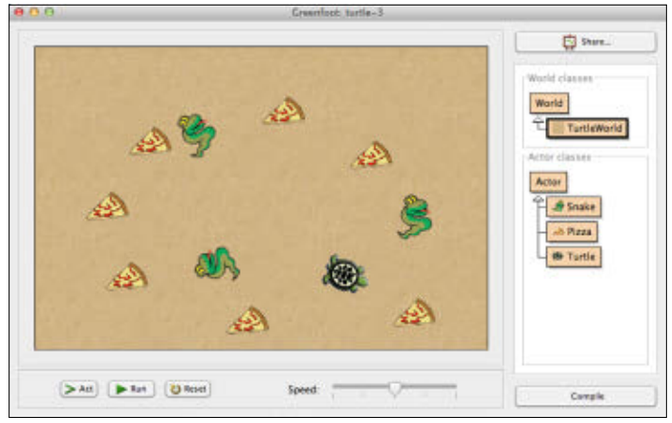


Figure 1

Checking for the left edge is fairly easy, because we know that the coordinate of the left edge is always zero. So we can write the following:

```
if (getX()==0) ...
```

In fact, we want to start turning not only when we hit the edge, but when we get near it. Let's say we define "near" as being within 10 pixels. So we can write the following:

```
if (getX() < 10) ...
```

That will work well. Checking for the right edge, however, is a little more complicated, because the coordinate of the right edge of a world can vary.

One option is to look into the source code of the TurtleWorld class to check the actual size of our world. If we do this, we can see that our world is 600 x 400 pixels in size, so the rightmost position is at *x* coordinate 599.

Adding again a 10-pixel buffer, we could now extend our if statement from above to also check for the right edge:

```
if (getX() < 10 || getX() >
589) ...
```

PHOTOGRAPH BY JOHN BLYTHE

The double vertical bar (||) is an or operator, so we can read this as follows:

```
if (x is less than 10 or x
is greater than 589) ...
```

This will work for this particular world, but only as long as we do not decide at some point to change the size of our world. A better solution would be to just call a method and ask the world how wide it is.

We can do this by changing our if statement to this:

```
if (getX() < 10 || getX() >
getWorld().getWidth()-10)  ...
```

In this condition, getWorld() calls a method to retrieve the world object itself. This object will be returned from this method call. We then call a second method, getWidth(), on this world object.

This technique is much better than hardcoding the actual width of the world into our class, because it will flexibly adapt to changes in world size.

The whole method, after we complete this if statement and we also add another one to check the *y* coordinate, is shown in **Listing 1**.

if/else Statements

Another new construct we see in **Listing 1** is an example of an if/else statement. We have seen if statements before, consisting of a condition and a body. The body was executed only when the condition was true.

This new form of the if statement has an additional keyword, else, and a second code block attached to it (see **Listing 1**). This second block is executed when the condition is *not* true.

Completing the Turn

We now have a method that allows us to check whether we are near the edge of the world. The next bit to do is to turn around if we are. This is now fairly easy. We add an if statement to our moveAndTurn() method, using atWorldEdge() as the condition and turning when we note that we are at the edge.

Listing 2 shows the new version of the moveAndTurn() method with this improvement. Try this out in your own scenario. You should note that the snakes now turn around very effectively once they get to the edge of the world.

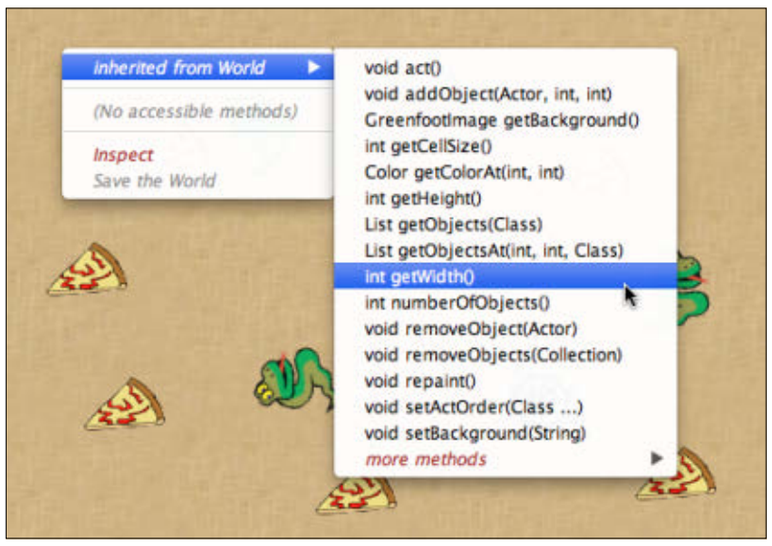


Figure 2

LISTING 1 LISTING 2

```
/**
 * Test if we are close to an edge of the world. Return true if we are.
 */
public boolean atWorldEdge()
{
  if(getX() < 10 || getX() > getWorld().getWidth() - 10) {
    return true;
  }
  if(getY() < 10 || getY() > getWorld().getHeight() - 10) {
    return true;
  }
  else {
    return false;
  }
}
```

Download all listings in this issue as text

An Important Lesson

The most important lesson we learn from this is this: The world is an object, too, and it has useful methods.

We can investigate the world methods in two ways: We can either just call them or we can read their documentation.

We can call a world method by right-clicking the world background and selecting a method from the Inherited from World submenu (see **Figure 2**).

You should try, for example, calling the getWidth() and numberOfObjects() methods to see what they do.

> **REMEMBER THIS**
> The most important lesson we learn is that **the world is an object**, too, and it has useful methods.

To investigate the world methods by reading their documentation, go to the main Greenfoot window and select **Greenfoot Class Documentation** from the **Help** menu. This opens a Web browser with the documentation for all six Greenfoot API classes. Select the World class and you can see a list of all available methods and their descriptions.

Adding Sound

Another nice thing we can do to improve our little game is to add sound effects. Included in the turtle-2 scenario (which you can download using the link at the beginning of this article) are two sound files named slurp .wav and au.wav. (You can

Figure 3

see them by looking in the sounds folder within the turtle-2 folder.)

The Greenfoot class has a static playSound method that takes the name of a sound file as a parameter and plays the sound. So we can play the slurp sound simply by writing the following:

```
Greenfoot.playSound("slurp.wav");
```

Try adding this statement to your turtle code, so that this sound is played when the turtle eats some pizza. **Listing 3** shows the resulting eat() method.

If this works, make a similar change to the Snake code, so that the snake plays the au.wav sound when it eats the turtle.

Recording Your Own Sounds

Adding sound effects by playing prerecorded sounds is easy, and it adds quite a bit of fun to the project. But we can make this even better by creating our own sounds.

While you can find free sound-effect libraries on the internet and use effects downloaded from there, by far the easiest (and most fun) way to add effects is by recording them yourself. You can do this on any computer with an attached microphone.

Greenfoot has a sound recorder built in that makes it really easy to do this.

Using the main menu, choose **Show sound recorder** from the **Controls** menu. This will bring up the sound recorder interface in a separate window (see **Figure 3**).

This sound recorder allows you to record you own voice, or any other sound effect, and trim it to remove unwanted leading or trailing noise or silence. You can then save the sound under a name of your choice. Note that Greenfoot will always attach a .wav suffix to your filename. This is important because you must use the full filename when using the sound in your code.

For example, if you specify "bark" as the name for your sound, Greenfoot will name the file bark.wav, and you can play it in your scenario by using the following statement in your source code:

```
Greenfoot.playSound("bark.wav");
```

LISTING 3

```
/**
 * Look for pizza and eat it if we see some.
 */
public void eat()
{
  Actor pizza = getOneIntersectingObject(Pizza.class);
  if(pizza != null) {
    getWorld().removeObject(pizza);
    Greenfoot.playSound("slurp.wav");
  }
}
```

Download all listings in this issue as text

Adding your own sound effects is an easy and very effective way to personalize your scenario and make it much more attractive and fun to play.

The Greenfoot Class

Here is a general lesson to take away from this: It pays to be familiar with the methods available in the Greenfoot class. Bring up the Greenfoot class documentation again (as described above, using the function from the **Help** menu) and have a look through the methods available in this class.

We have already used some of them (for random numbers, keyboard control, and sound), but there are others that you might find useful for further extending your scenario.

The Greenfoot API is not big, and reading the documentation is time well spent if you plan to work more with Greenfoot.

Tip: Teachers and instructors can find a lot of teaching material in the Greenroom educator community.

Conclusion

In this article, we encountered two fundamental aspects of Java and Greenfoot programming:

The world itself is a Java object. It has methods that you can call to interact with the world. You can use the Greenfoot API documentation to find out what methods exist.

Greenfoot can easily record and play back sounds. This makes game scenarios much more attractive. The method to play a sound is in the Greenfoot class. `</article>`

LEARN MORE

- "Getting Your Feet Wet" in *Java Magazine*
- "Starting Your First Computer Game" in *Java Magazine*
- Java SE 7 API

The New Java Developers

Meet JavaMan

Bruno Souza on the future Java developer, the Java/Brazil connection, and Java's continuing importance. **BY JANICE J. HEISS**

Bruno Souza, a Java developer, open source evangelist, and Java Champion, is founder and coordinator of SouJava (Java Technology Users Society) in Brazil and co-lead of the worldwide Java user groups (JUGs) community at Java.net. Known widely as "JavaMan," he has a personal passion for nurturing developer communities and works actively with Java open source communities and projects. Souza has extensive experience working on large projects in the government, finance, and service industries and also develops cloud-based systems using Java.

In May 2011, Souza was elected to represent SouJava as a member of the Executive Committee of the Java Community Process (JCP).

PHOTOGRAPHY BY BOB ADLER

Java Magazine: You have been giving talks with colleagues about the "future Java developer." Tell us about this.

Souza: Java developers can prepare for their own future by understanding that they have several freedoms now, and they can use those freedoms today to become a developer of the future.

The first freedom is the one software gives you. Anything you can think of, you can do with software—you are only limited by your imagination. And that has lots of benefits. The freedom that software offers allows you to create all kinds of things, businesses, ideas, and plans. And because software allows for all this freedom, it means that software is very complex—one of the most complex things that human beings have ever come up with. As a corollary, we have to accept the fact that software development is hard, and it's not going to become easy just because there is a new language, a new framework, or a new tool.

The second freedom is the freedom to learn—and build on what you've learned—that you're given by open source. Of course you can learn from others by reading their books or participating in proprietary software projects. But if you're a developer, with open source you can learn

Software development is more art than engineering, says Java developer and SouJava founder Bruno Souza. "If you're not creative and don't have imagination, it's very hard to be a great developer."

more by working with other developers, by looking at and studying the code created by them. So if you want to be a great developer, you can just find some good project using the technology you want to learn and the libraries you're interested in. And you can see how those projects were done, where they made mistakes, and where they succeeded. You can see where they have great performance and poor performance. You can learn from the code, experiment with it, build on top of what other people are doing, and have others building on top of what you did. It's like in the world of science where every new paper or new discovery builds on top of everything that was discovered before.

A third freedom is freedom from platforms. Java had freedom of platform as a core objective, and because Java was so successful at that, it has helped other languages with similar goals to succeed. And now most development technologies allow us to develop software independent of the platform that you have underneath. This has even promoted things like cloud computing, where you don't care about what hardware you're developing under and your applications can run anywhere.

Some vendors, however, are still tied to the idea that they're the only provider of the platform, and they're trying to lure developers by saying, "Hey, come to my platform because it's the only one that actually works." And the more the developer falls into this trap, the worse he or she will be prepared for the future. So there are a lot of technologies, products, frameworks, and so on that enable you to be vendor independent, platform independent. You should benefit from these for any application you develop.

A fourth important freedom is the freedom from hardware itself. Imagine that you have a 3-D printer in your home. You need a new chair, so just print a chair. You need new silverware—just print the silverware. All you need is to design and print whatever you need. That gives you a great deal of freedom. We're still not there yet; that's in the future. But as a software developer today, you have that freedom: you can design and create anything you want because software allows you to do that. And if you want to run this environment, you can run it in any way you want, because you have the ability to have as many servers, machines, environments, networks—as many of each thing you need to create your software today.

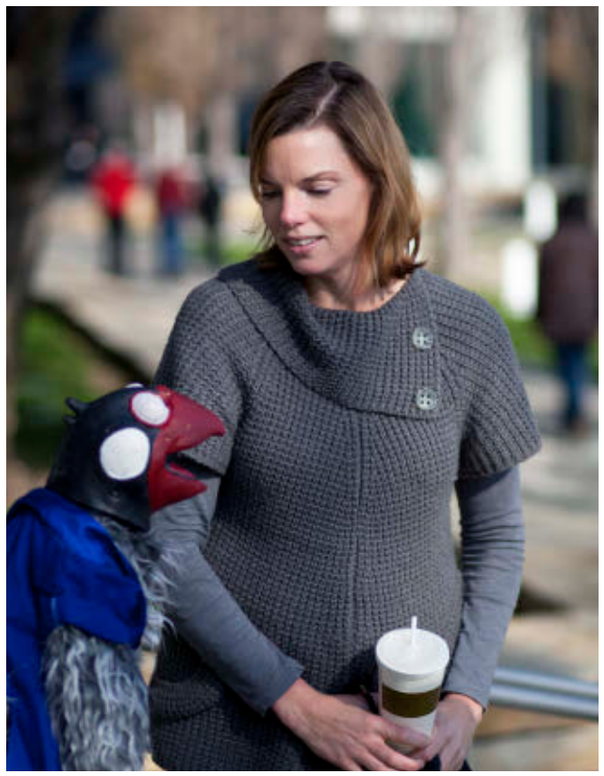
JUST BUILD IT

"As a software developer today, you have that freedom: you can design and create anything you want."

That's the freedom that cloud computing brings to you: it gives you the ability to have everything you need to run your software. It doesn't matter if you need one machine or a cluster of thousands of machines. Cloud computing gives you this freedom. It still has its problems, and you need to make sure to keep your freedom of vendors, but it is a great platform for the future.

The final freedom is the freedom to work from anywhere, because if you can work from anywhere, it also means you can work with anyone else from anywhere. That means your team doesn't need to be confined to one specific room or any one city or country. And

Bruno Souza stirs things up on the Oracle campus.



SouJava's Juggy chats with *Java Magazine*'s Caroline Kvitka about life as a JUG mascot.

instead of going after whoever is willing to work with you in your region, you can find the best developers in the world to work with you. That opens up the possibility of doing things that really matter. And you can learn from or work with anyone anywhere in the world.

So if you take all of these freedoms into account, you have a glimpse of what the future looks like. You know that the developer of the future will really benefit from all the open source code that's out there and do more with less effort. Developers have all the tools and connections to change the world.

Java Magazine: You're a puppeteer. What's the role of imagination in being a good developer?

Souza: Imagination and creativity give us the first freedom, the freedom to do anything we want. I believe that software development is much more art than engineering. If you're not creative and don't have imagination, it's very hard to be a great developer. So developers should find ways to promote their creativity and imagination. I work with puppets, both building them and also giving them life. A lot of developers do music—I would say that is very popular. Many people do other things, such as knitting—seriously, many developers knit. Others paint or draw. I think that creativity is very important. And the more creative things you do, the better you are at being creative.

Java Magazine: What are some keys to having a successful JUG?

Souza: A JUG is a great way to network. It consists of people who are socializing together to learn the technology. It's a place where we come to discuss technology that we like, learn from each other, share experiences, and even get jobs.

So it's very social—that's why I like to say that the JUG groups are the real Java community because they're the real people. JUG members can come from numerous backgrounds. We have very hard-core developers participating in the Java groups; we have students who are learning how to use the technology. People working from large corporations to small startups are also there. What makes JUGs successful is really the passion of the developers who are willing to organize the group.

HE'S THAT GUY

"I'm the guy who starts things. I like to be there in the beginning, believe in the original idea, make it happen, and create something new—and then just let someone else finish the work and maintain it for the long term."

Very few people are actually willing to spend the time to create the environment to make this happen—organize events, go after sponsors, and do the preparation work that allows the group to actually succeed. So I think the key to a successful JUG is in the passion of the leaders.

But leaders change; they are volunteers, and they get tired or busy. JUGs need to keep reinvigorating themselves with new people who are willing to do the work. We need new people to join who have fresh ideas and want to try out new things. The most-successful groups are passionate and have a great support group of people sharing the leadership and the work.

Java Magazine: Brazil not only has one of the liveliest and richest Java communities in the world, but the Brazilian government also performs some of the most innovative and constructive uses of Java. Can you tell us why Java and Brazilian culture are such a good fit?

Souza: Probably the most important reason is that the first companies to use Java in Brazil—some of them tied to the government—were looking for freedom from vendors. They'd had problems for many years working with vendors who were international companies and had to cope with situations where, in effect, one company controlled how you did business. Every time you developed an application, you were tying yourself to one particular vendor, and there was no way to avoid that. Java offered the promise of

Souza proudly wears the flag of Brazil, a country that embodies the spirit of Java.

COME TOGETHER

"A JUG is a great way to network. It consists of people who are socializing together to learn the technology. It's a place where we come to discuss technology that we like, learn from each other, share experiences, and even get jobs."

not tying you to a single vendor, a single platform, or a single development model. And the government and large companies were looking for this freedom. That's why Java is so successful in Brazil and worldwide.

Moreover, around the year 2000, the Brazilian government started to promote open source and free software, which was an important part of this search for freedom. Java was created from the start with a focus on building communities around it. So when Java came out in 1995, it came out with the source code and allowed people at universities, developers, companies, and others to actually see what was going on and participate in the creation and evolution of Java. Also, the JCP was created shortly after that. So Java was always more of a community-driven technology than a vendor-driven technology. It lined up with this idea of freedom that the companies were looking for, and with the idea of user groups and the community. There were a lot of Java user groups in Brazil from the beginning—we are a very social culture, so a community-driven technology fits well in Brazil.

All these things combined to push Java in Brazil, and for many years now, it's been the main development technology in the country.

Java Magazine: James Gosling once remarked that there were excellent developers throughout the world, but the craziest developers were definitely the Brazilians—he meant this as a compliment.

Souza: One thing about Brazilian culture—we like to do things on the fly and do whatever needs to be done to make something work. And we try to make things fun. The fact that we're willing to operate this way is one reason why Java is a good fit and why user groups are so important in Brazil. What's more, in Brazil we have often not had the resources to do everything we want so we learn to get by with the resources we have, which helps spur creativity.

We're also optimistic and unafraid to take risks—we assume things will work out. I've seen projects in Brazil that have won the Duke's Choice Award that might not have taken off elsewhere because of all the risks. Brazilian developers just went ahead instead of assessing all the risks involved.

Java Magazine: Where in the process of programming do you have the most fun?

Souza: I'm the guy who starts things. I like to be there in the beginning, believe in the original idea, make it happen, and create something new—and then just let someone else finish the work and maintain it for the long term. I enjoy the initial creation and development—and then someone else comes in and can fix my mistakes!

Java Magazine: What do you do when you get stumped?

Souza: When I get stuck, I just keep working and trying different things. I don't like to stop when I'm stuck. And I always like to rest when something is done. When I'm stuck, I feel like nothing is complete, so I keep working until I can see some solution. Then I can take a break and rest. I also like to discuss what's going on with someone. I'll try to explain what's wrong and why I can't go forward, and that helps me think more than anything else. Having someone take a look at what you're doing can usually bring the flaw to the surface. `</article>`

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

LEARN MORE

- Bruno Souza's blog
- SouJava

JULIEN PONGE

BIO

Part 2

Project Coin: The Java Language Has Evolved!

Java SE 7 offers several features for writing more-concise and safer code while not breaking backward compatibility with existing Java code.

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

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

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

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

This article focuses on the simplified varargs method invocation, the try-with-resources statement, the multi-catch, and the more precise rethrow.

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

Simplified varargs Method Invocation

Java SE 5 introduced generics. They have since been widely adopted because of the increased safety that the compile-time type checking offers. They are especially useful when dealing with collections, because the generic type declaration clearly documents its intended usage and removes unnecessary casts.

The implementation strategy for generics is a compromise between a complete type system and the need for backward compatibility with existing code bases at the time generics were introduced. To accomplish that, the compiler knows the parameterized types and performs checks based on them, but such information is lost at runtime. This is called *type erasure*.

To put it in other words, List<String> and List<Integer> are types enforced at compilation time, but they cannot be *reified* at runtime. The Java Virtual Machine (JVM) deals with the raw List type in each case.

Despite the increased type safety brought by generics, there

Author Julien Ponge provides an introduction to Part 2 of his Project Coin series.

PHOTOGRAPH BY MATT BOSTOCK/GETTY IMAGES

are easy means to introduce unsafe code. Indeed, consider the following method:

```
private static <T> void
doSomethingBad(List<T>
list, T... values) {
  values[0] = list.get(0);
}
```

It looks relatively harmless at first sight. It takes a list of generic type T and a variable list of arguments of type T, and it assigns the first element of them to the first one of the list. A variable arguments list is simply a primitive array in the common type of the elements. There is no need to cast the call to list.get(0) to type T because of the generic type concordance. Still, compiling code with such a method raises an *unchecked warning* caused by potential *heap pollution*.

To illustrate this, let's consider the main() method shown in **Listing 1** to invoke the doSomethingBad() method shown previously.

Formally, heap pollution happens when a variable of a parameterized type points to another one whose type is not of that parameterized type. In this example, list is of raw type List, which is not parameterized, but it is assigned with the return value of Arrays.asList(), which resolves to List<String> in that specific case.

Because the variable argument list is made of integers, the parametric type T in this invocation of doSomethingBad() resolves as Integer. Hence, the argument list is expected to be of type List<Integer>.

This is where the case of heap pollution happens. We could pass a type List when a List<Integer> was expected. Because of type erasure, however, the compiler could not check that further. Hence, the result is compilation with "just" a warning. If we run this program, as shown in **Listing 2**, we get an exception, because we cannot assign a String to an element of an array of Integer.

Of course, this would not have happened had we passed a genuine list of integers instead of a list of strings. The compiler raises unchecked warnings in any place that might yield to heap pollution. Note that heap pollution does not just happen with raw types. Managing to assign a variable of type List<String> to a value of type List<Integer> is also a case of heap pollution.

More generally, this warning is emitted whenever a method has a variable argument list of a type T that is not reifiable, because it can introduce unsafe operations. Types such as Object, String, or Integer are reifiable, while a type with a parametric type T, such as in the doSomethingBad() method, is not. Hence, the type of the variable argument list *values* of that method is T[], which is not reifiable.

There are, however, countless examples where variable argument lists and generics mix safely, because method implementations do not perform unsafe operations. Examples within the Java SE platform classes include java.util.Arrays.asList() and java.util.Collections.addAll(). Such methods only read the variable argument arrays and put the values into collections. Yet, calling those methods generates warnings, which developers often disabled before Java SE 7 by adding @SuppressWarnings({"unchecked", "varargs"}) annotations on the invoking methods. This practice adds boilerplate code, and it might remove warnings for truly unsafe actions inside such methods.

Now, consider the example in **Listing 3**.

The compiler generates a warning for possible heap pollution. However, looking at the implementation, we know that this code is safe. As the vendor of the method, as of Java SE 7, we can reflect this by adding the java.lang.SafeVarargs annotation shown in **Listing 4**. This is useful both for documentation purposes and for removing unnecessary warnings.

This annotation has been added in several places in the standard library classes, including java.util.Arrays.asList() and java.util.Collections.addAll(), although it should be noted that in the case of java.util.Arrays.asList(), the returned list is not a copy but a list backed with the array that is passed as an argument.

There are still restrictions on using @SafeVarargs. It can be applied only to *static* methods, *final* instance methods, and constructors, all having variable argument lists. Indeed, there is no point in applying @SafeVarargs on a method or constructor with a fixed arguments arity. Annotation inheritance works only with classes—not interfaces, instance methods, or constructors. This makes it impossible to substitute the safe code that a vendor provided and annotated with @SafeVarargs with a potentially unsafe one. In such cases, the overridden methods are not transitively declared to be safe unless their vendor explicitly decides to do so. This is why such methods should be either static or final.

Finally, compilers are encouraged to raise warnings when the variable argument list type is actually reifiable (for example, foo(String... args)) or when unsafe operations still happen in the annotated method, such as an

WELL BALANCED

The **impact of each Java SE 7 change** was remarkably balanced with a scientifically sound analysis of millions of lines of existing Java code.

LISTING 1 LISTING 2 / LISTING 3 / LISTING 4

```
public static void main(String[] args) {
  List list = Arrays.asList("foo", "bar", "baz");
  doSomethingBad(list, 1, 2, 3);
}
```

Download all listings in this issue as text

unchecked assignment of an element of the array.

This new annotation should especially appeal to the creators of APIs that take advantage of generics and variable argument lists.

try-with-Resources Statement

The typical Java application manipulates several types of resources, such as files, streams, sockets, or database connections. Such resources must be handled with great care, because they acquire system resources for their operations.

Correct practices for resources and exceptions management in Java have been well documented. For any resource that was successfully initialized, a corresponding invocation to its close() method is required. This requires a disciplined usage of try/catch/finally blocks to ensure that any execution path from a resource opening eventually reaches a call to a method that closes it.

Let's consider the method definition shown in **Listing 5**.

At first sight, this method does not do much harm. It opens a file called data and then writes an integer and a string. There is, however, a serious issue in this method regarding the call to the close() method. Indeed, suppose that an exception is thrown while writing the integer or the string because the underlying file system is full. Then, the close() method has no chance of being called.

BE CAREFUL
Resources must be handled with great care.

This is not so much of an issue regarding DataOutputStream, because it operates only on instances of OutputStream to encode and write primitive datatypes into arrays of bytes. The real problem is on FileOutputStream, because it internally holds an operating system resource on a file descriptor, which is freed only when close() is called. Hence, this method leaks resources.

Listing 6 shows a correct way to rewrite the method shown in **Listing 5**.

There is, admittedly, a lot of boilerplate code in this example to ensure that resources are properly closed. With more streams, network sockets, or JDBC connections, such boilerplate code makes it harder to read the actual business logic of a method.

The new try-with-resources statement introduced with Java SE 7 extends try blocks to declare resources similar to the case with for loops. Any resource declared within a try block opening will be closed. Hence, the new construct shields from pairing try blocks with corresponding finally blocks dedicated to proper resources management. A semicolon separates each resource, as shown in **Listing 7**.

Finally, such a try-with-resources statement may be followed by catch and finally blocks, just like regular try statements prior to Java SE 7. The correctWriting() method in **Listing 6** can be rewritten as shown in **Listing 8**.

A new interface called java.lang.AutoCloseable has been introduced in Java SE 7. All it does is provide a void method named close() that may throw a checked exception (java.lang.Exception). Any class willing to participate in try-with-resources statements should implement this interface.

It is strongly recommended that implementing classes and subinterfaces declare a more precise exception type than java.lang.Exception or, even better, no exception type at all if invoking close() should not fail. It has been retrofitted into many classes of the standard Java SE runtime that provide such close() methods, including the java.io, java.nio, java.security, java.sql, java.util.jar, java.util.zip, javax.crypto, and javax.net packages.

Now, let's suppose that we have an AutoClose class available that implements AutoCloseable. Let's use it as part of a try-with-resources statement:

```
public static void runInTWS()
throws MyException {
  try (AutoClose autoClose =
new AutoClose()) {
    autoClose.work();
  }
}
```

The code of runInTWS() is extracted, as shown in **Listing 9**, by the JD-GUI

LISTING 5 LISTING 6 / LISTING 7 / LISTING 8 / LISTING 9

```
private void incorrectWriting() throws IOException {
  DataOutputStream out = new DataOutputStream(new FileOutputStream("data"));
  out.writeInt(666);
  out.writeUTF("Hello");
  out.close();
}
```

Download all listings in this issue as text

decompiler tool (after reformatting and slight renaming of variables).

This is the syntax de-sugaring work performed by the Java compiler for try-with-resources statements. It generates safe and correct code to ensure that each initialized resource has its close() method eventually called.

It should be noted that Throwable objects support an addSuppressed() method that is new with Java SE 7. A common problem with resources management is that the close() method of a resource class might throw a further exception, thus masking the initial one. Because only one exception can be thrown at a time, an exception can now be attached as a "suppressed" exception to another one.

This is a novelty whose primary use case was that of having better exception handling for the try-with-resources statement, because resource management is highly prone to exception masking. The new getSuppressed() method of Throwable returns an array of Throwable. In turn, stack traces now have support for suppressed exceptions when invoking printStackTrace() on an instance of Throwable, yielding outputs such as the output shown in **Listing 10**.

In summary, the new try-with-resources statement not only makes code more readable, it also makes it much safer. For more information, see "Better Resource Management with Java SE 7: Beyond Syntactic Sugar."

MAKE AN EXCEPTION

A **common problem** with resources management is that the close() method of a resource class might **throw a further exception**, thus masking the initial one.

Multi-Catch and More-Precise Rethrow

The following changes regarding exception handling in Java SE 7 are significant. The most immediate one is the ability to group several catch clauses as one. Developers are often faced with portions of code where multiple checked exceptions can be thrown. This requires multiple catch clauses and arguably adds some redundancy in the source code, as we can see in **Listing 11**.

Listing 11 is based around some *reflection* in Java. The Class for String is loaded, an instance is created, and its toString() method is called through reflection. As we can see, there are five checked exceptions that can be thrown from this code. In reality, more exception types could be caught as several unchecked exceptions can be thrown from the reflective code, including SecurityException and NullPointerException.

Facing such a deluge of catch clauses, many Java developers would be tempted to simplify the code from **Listing 11** as shown in **Listing 12**.

This is a common pattern in exceptions handling code where developers take a general exception type, such as Throwable or Exception, and condense the error handling logic within a single catch clause. Indeed, all each clause does is call printStackTrace(), which is common to all exception types. This practice is nevertheless discouraged, because catching an overly general exception forbids a fine-grained, type-specific handling. Also, other exceptions might be accidentally caught beyond the developer's original intention.

In the first version of the code (in **Listing 11**), a SecurityException would be propagated to the invoker. In the second version (in **Listing 12**) with a sole catch (Throwable t) clause, SecurityException is caught and execution resumes normally. The application state might, however, be inconsistent. Hence, it is advised to precisely catch exceptions or let them propagate.

Starting with Java SE 7, catch clauses can be condensed using *multi-catch*, as shown in **Listing 13**.

The apparent benefit is that of factoring common exception-handling logic per group of exception types while keeping control of the exception flow. More formally, the type of an exception in a multi-catch statement must be a *union of alternatives*, where each type is separated by a |, and each type is a distinct subclass of Throwable with no inheritance relationship.

For example, ClassNotFoundException | InstantiationException is a union of alternatives because neither is a subtype of the other. By contrast,

LISTING 10 LISTING 11 LISTING 12 LISTING 13

```
MyException: Exception in work()
  at AutoClose.work(AutoClose.java:11)
  at AutoClose.main(AutoClose.java:16)
  Suppressed: java.lang.RuntimeException: Exception in close()
    at AutoClose.close(AutoClose.java:6)
    at AutoClose.main(AutoClose.java:17)
```

Download all listings in this issue as text

ClassNotFoundException | Exception is *not* a union of alternatives and is not allowed in a multi-catch statement. Similarly, ClassNotFoundException | ClassNotFoundException is also disallowed, because a type is also a subtype of itself.

A multi-catch statement is semantically equivalent to as many catch clauses as types take part in a union type where the exception reference is implicitly final. Our previous example is semantically equivalent to the code shown in **Listing 14**.

Compilers are not required to perform this code expansion. Given that each block is required to perform operations that are supported by the union of all alternative exception types, a single effective catch block is compiled per multi-catch statement.

This is easy to check by decompiling the code for our example, as shown in **Listing 15**, because we see that the exception table has jumps to the same address.

We mentioned earlier that exceptions of a union of alternatives are implicitly considered to be final. There is more to it, though, because any exception that is not used as a left-hand operand to an assignment is now considered to be *effectively final*. This means that the exception e in the following code is now considered to be final, although it has not been declared so:

```
catch (ClassNotFoundException e) {
 e.printStackTrace();
}
```

By contrast, it is not considered to be effectively final in the following contrived example:

```
catch (ClassNotFoundException e) {
 e.printStackTrace();
 e = new ClassNotFoundException();
 throw e;
}
```

It is perfectly legal to declare as final an exception reference in a multi-catch clause. It is, however, discouraged as a matter of style given that this is implicitly the case.

Now is the time for questioning why implicitly and effectively final exceptions are so important. The reason lies in what happens when an exception is being rethrown from a catch or multi-catch. Exception tables work with the actual type of an exception at runtime. The compiler needs to have a very precise understanding of the exception types being captured and thrown, especially when a union of alternative exception types is involved, because they do not correspond to reifiable types in the JVM. Imprecise information can lead to the generation of nondeterministic exception tables, where a given exception can match multiple entries in a table at runtime. It also leads to exception tables with dead-code entries. Let's consider the code shown in **Listing 16**.

This code compiles under Java SE 6, although SomeChildException and SomeOtherChildException are not subtypes one of the other. Under Java SE 6 semantics, the compiler checks that a throw expression of some exception type E actually throws an exception of type E, a subtype of E, or any type that can be thrown by a corresponding try block in a catch block. Then, each catch clause must be a type, subtype, or supertype of

LISTING 14 LISTING 15

```
try {
   Class<?> stringClass = Class.forName("java.lang.String");
   Object instance = stringClass.newInstance();
   Method toStringMethod = stringClass.getMethod("toString");
   System.out.println(toStringMethod.invoke(instance));
} catch (final ClassNotFoundException e) {
   e.printStackTrace();
} catch (final InstantiationException e) {
   e.printStackTrace();
} catch (final IllegalAccessException e) {
   e.printStackTrace();
} catch (final NoSuchMethodException e) {
   e.printStackTrace();
} catch (final InvocationTargetException e) {
   e.printStackTrace();
}
```

Download all listings in this issue as text

the possibly thrown exception types of the corresponding try block.

Here, the nested try block is understood to be throwing an exception of type SomeRootException, but this is imprecise, because in reality, it is of type SomeChildException. Nevertheless, the catch block on SomeOtherChildException is allowed under those semantics.

By contrast, the code in **Listing 16** does not compile with Java SE 7 because a precise analysis of the rethrown types is performed, as shown in **Listing 17**.

The exception type analysis works as follows:

- The first try blocks can throw an exception only of type SomeChildException.
- firstException is effectively final because it is not modified in the catch block.
- The compiler knows that firstException is actually of type SomeChildException, thanks to its being final.
- When firstException is rethrown, the compiler is aware that it is precisely of type SomeChildException and, hence, the try block is understood to throw only SomeChildException exceptions just like the topmost try block.
- The catch(SomeOtherChildException secondException) expression is invalid because no corresponding try blocks can throw it or a subtype of it.

The introduction of the multi-catch statement is not just beneficial for the purpose of making the Java language more concise, because its implementation required improving the exception type analysis.

LISTING 16 LISTING 17

```
public class Imprecise {

  static class SomeRootException extends Exception { }
  static class SomeChildException extends SomeRootException { }
  static class SomeOtherChildException extends SomeRootException { }

  public static void main(String... args) throws Throwable {
    try {
      throw new SomeChildException();
    } catch (SomeRootException firstException) {
      try {
        throw firstException;
      } catch (SomeOtherChildException secondException) {
        System.out.println("I got you!");
      }
    }
  }
}
```

Download all listings in this issue as text

Nevertheless, this change can break the compilation of existing source code, because the more precise exception type analysis can report errors as in the example provided in **Listing 17**. The decision to "go for it" was taken after examining millions of lines of existing Java code and realizing that such settings were extremely rare.

Conclusion

This article concludes a two-part series dedicated to describing the evolution of the Java language in Java SE 7. The changes allow writing more-concise and safer code while not breaking backward compatibility with existing Java code. Because the changes are subtle yet appealing touches to the language, rather than massive changes, we can expect them to be a strong argument in favor of a rapid adoption of the Java SE 7 platform. `</article>`

LEARN MORE

- Project Coin mailing-list archives
- Project Coin blogs
- Latest maintenance review of the Java Language Specification
- "Language Designer's Notebook: Quantitative Language Design"
- "Project Coin: Inducing Contributory Heap Pollution"
- "Project Coin: Safe Varargs"
- "Project Coin: Improving Multi-Catch and More Precise Rethrow"

Bring the Web to Your App

JavaFX WebView makes applications do new and interesting things via embedded HTML, CSS, and JavaScript.

JOSH MARINACCI

BIO

JavaFX 2.0 promises to reinvent how we create rich client applications. The new features and graphical back end let developers create amazing interfaces that are both attractive and responsive. One of the best new features of JavaFX 2.0 is the WebView control. The WebView lets you embed real HTML content in your application, opening up a whole new world of collaboration between the desktop and the Web.

The JavaFX team wanted to let developers embed HTML directly in applications with the richness and accuracy of a real Web browser, something the old Swing JEditorPane never achieved. To do this, they started with WebKit, the open source HTML rendering engine that is fast becoming the standard renderer for desktop and mobile platforms everywhere. It is used in Safari, Chrome, iOS, Android, webOS, and even the Amazon Kindle.

The JavaFX team also wanted to make the new Web control be more than just a simple wrapper around the native WebKit library. The WebView actually uses Java for onscreen drawing as well as all network access, opening up the door for much closer integration than would have been possible with a simple wrapper. You can even draw a WebView into a 3-D scene, as this rotated Web page example shows.

Ways You Can Use the WebView Control

Using the WebView in an application is very simple. Just create a WebView Java object and then get a reference to its WebEngine, which is the Java object that actually connects to WebKit. You can load a page by calling engine.load() with the URL of a Web page. You can load internally generated content directly from a string using engine.loadContent(). Then put your WebView in a JavaFX scene just like any other control. That's all there is to it.

Of course, if that were the extent of the WebView API, then we wouldn't need this article. Once you have the ability to bring Web content into your app, what can you actually *do* with it? I think there are four main use cases:

- Embed some part of the Web in your desktop app. Embedding Google maps in your app is a great example.
- Manipulate remote Web content using Java. This means you load content from the Web but modify it in some way using your Java code.
- View internally generated content or content loaded locally. Generating reports on the fly is a good example of this.
- Manipulate Java content from the JavaScript side. The WebView does not provide a way for JavaScript to contact the Java side, but with a bit of cleverness, we can do some interesting things.

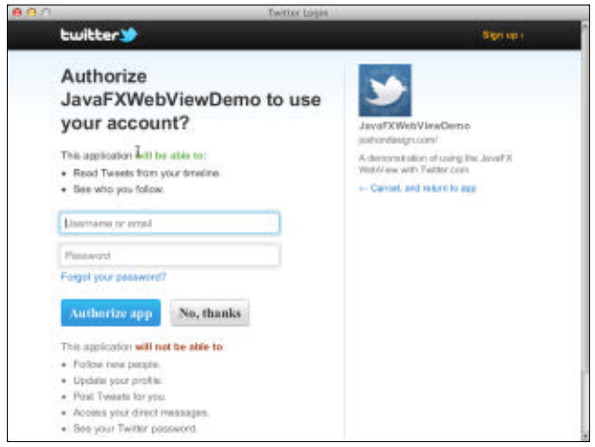


Figure 1

Reading and Embedding Web Content

Let's start with the first use case: embedding some part of the Web in your desktop app. Rather than simply wrapping a Web page, I thought it would be nice to take care of an issue that really annoys me: Twitter authentication.

Twitter uses OAuth to authenticate third-party apps. OAuth requires the application to send the user to a special URL on Twitter.com. The user approves the app there (see **Figure 1**) and is redirected back to the app that is requesting access. This works fine if the app is a Website, but if the app is a desktop program, there is no server URL for the user to be redirected back to. To address this issue, Twitter instead gives the user a pin code to type into the

PHOTOGRAPH BY CHRIS PIETSCH/GETTY IMAGES

Figure 2

app. This method works, but it's highly annoying and cumbersome. Often, the user gets lost between all the flipping screens and page redirections.

To solve this problem, we will use the WebView. Rather than opening a new Web browser, which might make the user lose context, we can embed the Twitter page *directly in our app*. Once the user approves the app, we can grab the pin code out of the document without requiring the user to type it in manually. We can do this by using the WebView.document property. Here's how it works.

First, you need a Twitter library to do the OAuth authentication. While it is possible to do everything by hand, you would have to deal with encryption and hash code generation, which is very annoying and error-prone. I am using the open source Twitter4J library, which takes care of these details for you (see **Listing 1**).

First, we must initialize Twitter4J using the consumer key and consumer secret specific to your app. You can get these values from your app configuration page on dev.twitter.com. Then, request the authorization URL. This is the URL that we must show to users so they can approve the app. Right now, we are on the main thread, but all JavaFX controls must be accessed on the GUI thread, so we call Application.launch() to open up a window on the right thread, as shown in **Listing 2**.

Opening the URL is simple. Create a WebView, get the engine, and then call load(). Then, we want to know when the user has typed in the username and password and pressed the OK button to get the pin. We can do this by adding a listener to the document property of the engine, as shown in **Listing 3**.

When the user logs in and approves the app, we will get a page like **Figure 2** that has markup similar to the following, somewhere in the middle of it:

```
<div id="oauth_pin">
<code>12334928</code></div>
```

The grabPin method is what will process the actual document. The document from the WebEngine is an actual W3C Document Object Model (DOM) document object, so we can use the standard APIs such as getElementById() to parse it. In this case, we want to look for the oauth_pin element, which is a DIV containing the pin that was given to the user. We want the text contents of the code element inside of it (see **Listing 4**).

Notice that I am returning early if the document is null or if the OAuth pin is missing. This is because the WebEngine document property might change several times: first when the app starts, then when the initial page is loaded, and then when we get to the pin page. We care only about the pin page, so the code safely does nothing in the other cases.

Once we have the pin text, we can complete the Twitter authorization process and request information about the user that only an approved app could get. In this case, I'm asking for the number of friends, as shown in **Listing 5**.

So, you can see that embedding and reading Web content in your app is quite easy. Listen for document changes and then read the contents of the page using the standard DOM API.

Changing Web Content

Now, what if we wanted to actually *change* the contents of the page rather than merely read it? The WebView lets you change the DOM through the returned Document object, or you can inject JavaScript into the page to manipulate the DOM.

For the next example, I will load up a Web page and then use the injection method to change its styling on the fly so the page will be more readable (see **Listing 6**).

The steps are the same as before: load up a page and add a document listener. In this case, we will load my tech blog, joshondesign.com, which has a nice design but can be a bit fancy for reading long amounts of text. It would be nice to change the page to a "reading" mode that uses a plain font and black and white. This would also be good for printing. The modifyDoc

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

```
public static void main(String ... args) throws Exception {

    //connect to twitter and get an authorization URL
    twitter = new TwitterFactory().getInstance();
    twitter.setOAuthConsumer(consumerKey,consumerSecret);
    requestToken = twitter.getOAuthRequestToken();
    authURL = requestToken.getAuthorizationURL();
...
```

Download all listings in this issue as text

COOL STUFF

The JavaFX 2.0 new features and graphical back end **let developers create amazing interfaces** that are both attractive and responsive.

method does this by injecting some JavaScript that will modify the style of the page's body (see **Listing 7**).

We can inject only a string, so first I put all the code into a StringBuffer. (A more robust version of this might load the JavaScript from a file instead of creating it in code.) Once we have the script, we can inject it by calling engine.executeScript(). This will return an object wrapping the return value of the code, which in this case is the bodys object. It is possible to use this return value to pass simple data from the JavaScript side to the Java side, but in this case, I just care that the code executed successfully.

Figure 3 shows what my blog looks like.

And **Figure 4** shows what my blog looks like with the new style.

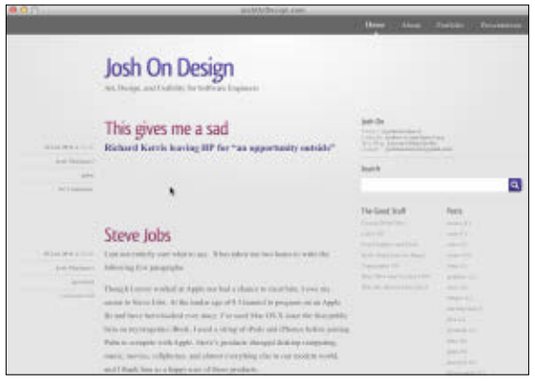


Figure 3

Figure 4

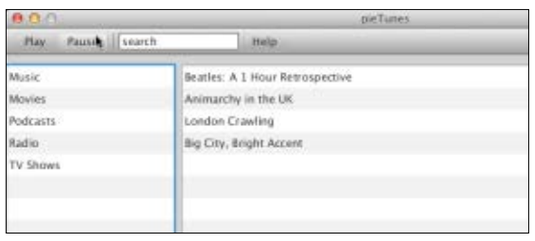


Figure 5

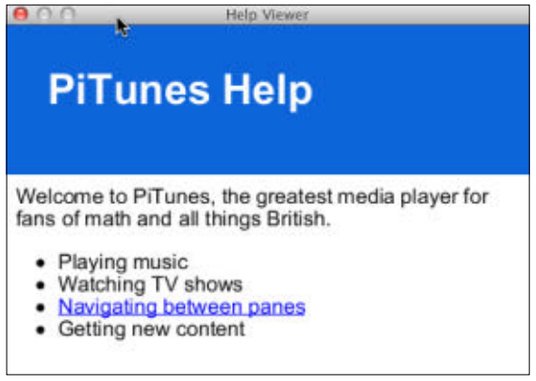


Figure 6

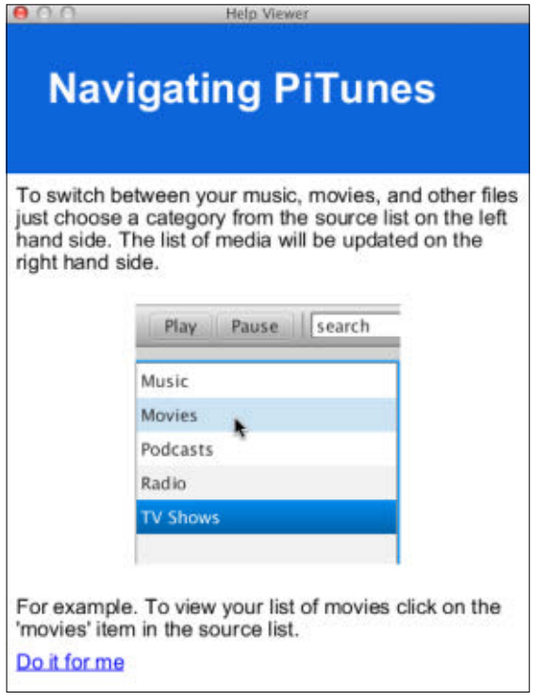


Figure 7

LISTING 7 LISTING 8

```
private void modifyDoc(Document newDoc, WebEngine engine) {
  StringBuffer script = new StringBuffer();
  //grab the body
  script.append("var bodys = document.getElementsByTagName('body');");
  //change the colors
  script.append("bodys[0].style.backgroundColor = '#FFFFFF';");
  script.append("bodys[0].style.color = '#000000';");
  //set a new default font
  script.append("bodys[0].style.fontFamily = 'sans-serif';");

  script.append("bodys;");
  //execute
  Object retval = engine.executeScript(script.toString());
  //return value doesn't matter in this case
  p("return value = " + retval);
  JSObject obj = (JSObject) retval;
}
```

Download all listings in this issue as text

Working with Generated Content

For the final example, I'd like to use the WebView to show some content that comes from the app itself: a help system. In days gone by, you would have to use a proprietary help engine to display rich text help within your app. Now, we can just write it all as HTML and show it in a window.

For this example, I created an imaginary media player called PiTunes. It has a toolbar at the top, a list of media types on the left side (called the source list), and a list of media files on the right, as shown in **Figure 5**.

To see the list of movies, the user clicks the **Movies** item in the list. This might not be obvious, however, so we can add some online help to explain. When the user clicks the Help button, a second window is opened with the help content. See **Listing 8**.

There are two important things to notice here. First, I am loading the help file from a URL that comes from getResource(). This means we can load content from inside a JAR file, not just on disk. Second, if you click the link from the first page to the second, you will see an image. This is a PNG file inside the resources package. The markup just references it locally: <img src="screenshot.png">. Because the WebView uses Java for all resource loading, we can load content from anywhere, even Java-specific locations, such as inside JAR files or databases. Everything just works. See **Figure 6** and **Figure 7**.

Now that we have rich help content inside our app, we can take it one step

LISTING 9

```
engine.setOnAlert(new EventHandler<WebEvent<String>>() {
  @Override
  public void handle(WebEvent<String> t) {
    sourceList.getSelectionModel().select(1);
  }
});
```

Download all listings in this issue as text

further. Instead of telling the user how to do something, we could actually do it for them. We could have a link that makes the app perform the action that the help text is talking about.

Of course, doing this sort of automation requires code on the JavaScript side to be able to access code on the Java side, which isn't possible yet. (Such a feature might come in a future release of JavaFX.) However, there is a clever workaround. JavaScript code can call the standard alert() function. The WebEngine lets you create a callback handler to deal with alert events. Rather than opening up an actual alert dialog, we can use this function as a way for the JavaScript side to communicate with the Java side. In this case, we just want to know when the user clicked the link; then we can update the GUI.

On the markup side, we just need a link that has a JavaScript URL:

```
<a href="javascript:alert
('movies')">Do it for me</a>
```

On the Java side, I created a handler for the alert event, as shown in **Listing 9**.

Now, when the user clicks the link, the GUI will change the selection to the second row (Movies). The user can both read and *see* how the application works, with almost no extra coding.

RETHINK RICH APPS

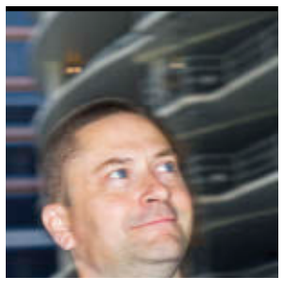
JavaFX 2.0 promises **to reinvent how we create** rich client applications.

Conclusion

HTML, CSS, and JavaScript are very powerful technologies, and now we can embed these technologies in our Java apps to make them do new and interesting things. This article just barely touches on the possibilities. A few other things you might want to try include embedding an RSS reader, caching Web pages for offline reading, or creating an entirely Web-based interface that runs locally as a double-clickable Java app. The possibilities really are endless. `</article>`

LEARN MORE

- JavaFX
- Josh on Design blog



SIMON RITTER

BIO

Part 3

JavaFX and Swing Integration

Use JavaFX to create visual effects for a more exciting toolbar.

This is the third and final article in a series about integrating JavaFX with existing Swing applications. In Part 1, we looked at the basics of wrapping a JavaFX scene as a JComponent and how to handle events. In Part 2, we looked at the use of a JavaFX table in a sample application. Here, we'll use visual effects to create a more exciting toolbar for the same sample application, which delivers details about stock quotes.

Note: Download the code for the sample application here.

The application includes a toolbar that uses icons to provide quick access to commands such as open a portfolio, refresh stock prices, and create a new portfolio. This functionality makes use of the Swing JToolbar component. To make the toolbar more visually interesting, we'll replace it with the ToolBar control from JavaFX, retaining all the same functionality.

Simon Ritter provides highlights from his three-part article series on JavaFX and Swing.

PHOTOGRAPH BY BOB ADLER

Adding Functionality

Let's start with the JavaFX code first and then see how we integrate this code into our application. To give the icons more functionality, we'll create a new class, ActiveIcon. The outline of the constructor is shown in **Listing 1**.

The code in **Listing 1** simply takes the name of an image passed to the constructor and creates an ImageView node that can be used on the toolbar. We also need the width of the image for one of the effects we will produce. As you can see, we've defined four types of effects that the icon can have: fade, spin, grow, and slide.

The implementation of these effects is shown in **Listing 2** and **Listing 3**, which is also part of the constructor.

The visual effects change properties of the icon using a Timeline. Each of the properties we're using is a DoubleProperty, which we can access via the appropriate method call to the ImageView object. Once we have a property, we create a Timeline that has one KeyFrame at zero seconds and another KeyFrame at 1,000 ms (one second). Each KeyFrame uses KeyValue to define the associated properties that are to be changed and the values for the properties at the given point on the Timeline. To make the effect continuous, we set the Timeline to be auto-reversing and have an indefinite repeat count. Implicit binding is used between the icon properties and the value being changed in the timeline.

The fade effect varies the opacity of the icon, the grow effect varies the scaling in the *x* axis and the *y* axis, and the spin effect rotates the icon through 360 degrees.

The slide effect is slightly more complicated because we move the icon along the *y* axis by twice the width of the icon. To make this fit with the other icons, we need to place an invisible rectangle behind the icon to provide enough space. We do this by setting the opacity of the rectangle to zero. (Note that this would not work by setting the visibility property to false).

Constructing the New Toolbar

Using the ActiveIcon class, we can now construct our toolbar, as shown in **Listing 4**. The code in **Listing 4** adds a new method to the StocksMonitorMainWindow class, which is responsible for constructing the main GUI. Here, we use the same technique described in the first article to

create a JFXPanel that wraps the JavaFX nodes and can be manipulated in a Swing GUI in the same way as any other Component object.

We construct a new, final instance of JFXPanel and then set up a task to run on the JavaFX thread to construct the toolbar node. It is very important that all manipulation of the JavaFX scene be handled in this way.

Within the run method, we create each element of the toolbar as a Button control. To match the original format, we use only a graphical icon (implemented with ActiveIcon) with no text. Adding tooltip text is a simple method call.

To respond to the user clicking a button, we add a new event handler to each button. Again, within the handler method it is important that all activities happen on the Swing event thread, so we add a Runnable object via the SwingUtilities invokeLater method. All the run method has to do is invoke the appropriate actionPerformed method that would be called from the Swing toolbar via registered listeners. Strictly speaking, we should pass valid ActionEvent objects to these methods, but none of the methods use the argument, so for brevity of code we pass null.

The toolbar itself is constructed by passing references to the buttons we've created, along with any required separators, to the constructor. The items in a toolbar are typically Button, ToggleButton, and Separator objects, but any Node can be used. So, it would be simple to replace a static separator with a more dynamic, custom version.

Replacing the Old Toolbar

The last piece is to replace the instance of the JToolbar with our new JavaFX version. **Listing 5** shows the changes to the initMainGUI method.

In order for the preferences to work (and to avoid a null-pointer exception), we must also change the synchGuiWithGeneralLookPrefs method, as shown in **Listing 6**.

Conclusion

In this series of articles, we've looked at how to integrate JavaFX scene graphs into a Swing application by wrapping the scene graph as a Component, which can easily be manipulated by a layout manager. We've also looked at how events can be handled using both the Swing event dispatch thread and the JavaFX application thread. The real benefit of having this kind of integration is that you can gradually migrate an existing Swing application to JavaFX without having to rewrite it all in one go. As the examples have shown, it is straightforward to replace tables, toolbars, and other standard components and have these new components add rich, visually appealing effects such as translucency and animations.

JavaFX is an incredibly rich and powerful way to develop exciting new user interfaces in Java. Get started! `</article>`

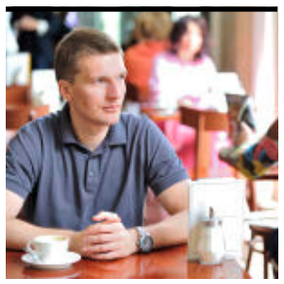
LEARN MORE

- "JavaFX and Swing Integration, Part 1"
- "JavaFX and Swing Integration, Part 2"
- Documentation and tutorials

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

```
public class ActiveIcon extends Parent {
 public static final int TYPE_NONE = 0;
 public static final int TYPE_FADE = 1;
 public static final int TYPE_SPIN = 2;
 public static final int TYPE_GROW = 3;
 public static final int TYPE_SLIDE = 4;

 public ActiveIcon(String fileName, int type) {
  URL imageURL = getClass().getResource(fileName);

  if (imageURL == null) {
   System.out.println("NULL image URL");
   return;
  }

  Image iconImage = new Image(imageURL.toString());
  ImageView iconView = new ImageView(iconImage);
  double width = iconImage.getWidth();

  ...

  getChildren().add(iconView);
 }
}
```

Download all listings in this issue as text



Who Needs Aspect-Oriented Programming?

In Java EE, there is rarely an obvious use case for AOP.

ADAM BIEN

BIO

According to Wikipedia, *aspect-oriented programming* aims to separate reusable functionality from the business logic: "In computing, aspect-oriented programming (AOP) is a programming paradigm which aims to increase modularity by allowing the separation of cross-cutting concerns. AOP forms a basis for aspect-oriented software development."

AOP was a hot topic a few years ago, but it never really took off in J2EE—and for good reason.

Note: A sample Java EE project for this article (AOPandJavaEE) is available on the Project Kenai Web page.

> **NOT SO HOT**
> AOP was a hot topic a few years ago, but **it never really took off in J2EE**—and for good reason.

Aspects Are Commodities

The integral parts of the Enterprise JavaBeans (EJB) 1.0 specification (JSR 318) in early 1998 were aspects. The developer had to develop the Bean class and the Remote and Home interfaces but none of the cross-cutting aspects. Threading, pooling, security, transactions, session handling, and even persistence were configurable aspects cleanly separated from the business logic. The cross-cutting functionality was configurable in the early EJB 1.0 days with POJOs.

With the advent of EJB 1.1 and J2EE, Java classes were replaced with XML, and the recent Java EE 5 and Java EE 6 specifications made XML optional. You can configure the vast majority of all use cases with annotations and override them with XML.

Although the whole EJB 1.0 idea was based on aspects, you will find only two mentions of the word *aspect* in the original specification in different contexts. AOP and EJB 1.0 were "invented" nearly at the same time.

In J2EE, the configurable decoration of an EJB bean with, for example, transactions was just an integral part of the development experience. Other frameworks provided the same functionality but called that functionality AOP.

You will have a number of aspects by defining a simple EJB 3.1 bean. For example, a stateless EJB 3.1 bean comes with transaction and lifecycle aspects out of the box.

```
@Stateless public class Greeter {
  public String greet(){
    return "Morning!";
  }
}
```

The method greet() is never invoked directly. The container dispatches the invocations to free EJB instances from the pool. Also, transactions are started before each method invocation. The method call is processed by a chain of default and configurable aspects until it reaches the actual business logic. Outside the application server, you would probably implement the same functionality with the decorator pattern or with a more generic solution, such as AOP.

Generic Decorators: Interceptors

Although the Greeter EJB bean is injected directly into a Servlet, as shown in **Listing 1**, you can still decorate the call.

An interceptor is similar to the Proxy class introduced with JDK 1.3, but it is easier to use. You only have to implement a class with a single method. **Listing 2** shows a generic business method interceptor.

Although the method name can be freely chosen, the return type, the exception, and the parameter have to match the signature defined in **Listing 2**. An interceptor is associated with a given

PHOTOGRAPH BY THOMAS EINBERGER/ GETTY IMAGES

managed bean or EJB bean with the @Interceptors annotation.

```
@Stateless
@Interceptors
(NicenessExtender.class)
public class Greeter {}
```

The value of the @Interceptors annotation is a class array. The order of the declaration defines the order of interceptor execution. Although this approach makes the Greeter EJB bean strongly dependent on NicenessExtender, in most cases, this approach is the best choice. There is no surprise; the decoration is obvious.

In all modern IDEs, you can also easily navigate from the EJB bean to the aspect for the price of direct binary coupling.

A declaration on the class level causes the interception of all public methods. A method annotated with @ExcludeClassInterceptors will not be intercepted by an interceptor defined on the class level. Similarly, the @ExcludeDefaultInterceptors annotation deactivates interceptors defined in XML.

A default interceptor, which is applied on all EJB beans, has to be defined in the ejb-jar.xml deployment descriptor (see **Listing 3**) and placed in the [WAR]/WEB-INF directory.

Interceptor declaration in the ejb-jar.xml file is not only limited to default interceptors. You can specify the bean and method names and have fine-grained control over the interception of methods. You can even override the annotations in the ejb-jar .xml file, which is useful for staging. Production settings can be easily overruled with ejb-jar.xml configuration. As the name of the ejb-jar.xml file implies, only EJB beans can be configured with the ejb-jar.xml file.

LOCATING POINTCUTS

In AOP, you can use **regular expressions, custom languages, and APIs** to locate the pointcuts, whereas in Java EE 6 you are usually relying on annotations or XML deployment descriptors to control the interception.

From Annotations to XML and Back

Greater decoupling and flexibility come with the @InterceptorBinding meta-annotation. **Listing 4** shows a custom @InterceptorBinding with an attribute.

With @InterceptorBinding, you can create your own annotations and use them to control the interception. The principle is similar to @Qualifier; the custom annotation denoted with @InterceptorBinding is applied on the interceptor, as well as on the intercepted class. The annotation itself and also its internal elements are considered in the resolution process.

With an additional element inside the annotation, you can even switch between multiple interceptors. See **Listing 5**, which shows the USUAL interceptor.

The previously introduced NicenessExtender will intercept all classes denoted with the @Nice(USUAL) annotation,

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

```
@WebServlet(name = "GreetingController", urlPatterns = {"/GreetingController"})
public class GreetingController extends HttpServlet {
  @Inject
  Greeter greeter;

  @Override
  public void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {
      out.println("<html>");
      out.println("<body>");
      out.println("<h1> " + greeter.greet() + "</h1>");
      out.println("</body>");
      out.println("</html>");
    } finally {
      out.close();
    }
  }
}
```

Download all listings in this issue as text

and the PersonalNicenessExtender will intercept all managed beans and EJB beans with the @Nice(PERSONAL) annotation (as shown in **Listing 6**).

You can choose between the interceptors by setting the proper value of the Level enum from the @Nice annotation. Here is an example of applying the annotation on Greeter:

```
@Nice(PERSONAL)
public class Greeter {}
```

It is not sufficient to declare the interceptors; they also must be activated. Each interceptor has to be declared in the beans.xml configuration file for activation. **Listing 7** shows the interceptor activation in WEB-INF/beans.xml.

The order of declaration in the beans .xml descriptor also specifies the order of decoration.

With @InterceptorBinding, the intercepted class is fully decoupled from the interceptor class. The interceptor and the intercepted class are dependent only on the @InterceptorBinding annotation. The annotation is the link between both parts.

Stereotypical Interceptors

To access the Greeter from JavaServer Faces (JSF), you will have to make it

available in the JSF Expression Language (EL) with the @Named annotation and define a scope such as @RequestScoped. Here is an example of using explicit annotations for JSF integration:

```
@Named
@RequestScoped
@Nice(PERSONAL)
public class Greeter {}
```

The InterceptorBinding annotation can be combined with stereotypes, which greatly reduces the amount of annotation declaration. Here is a decorated presenter stereotype:

```
@Model
@Nice(PERSONAL)
@Stereotype
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
public @interface NicePresenter {}
```

@Stereotype works like a macro. All annotations placed on @Stereotype are going to be expanded. @Stereotype is just an annotation container without any additional semantics. The expansion is recursive; a stereotype can be annotated with other stereotypes. **Listing 8** shows the built-in @Stereotype javax.enterprise.inject.Model.

The NicePresenter shown above was annotated with @Model, which in turn is a built-in stereotype containing the @Named and @RequestScoped annotations (see **Listing 8**). Also, in the case of stereotypes, you have to activate the interceptors in the beans.xml file.

Separation of Cross-Cutting Concerns

The majority of the interceptor code deals with generic invocations and reflection. The parameter javax.interceptor.InvocationContext carries the target method, its parameters, and additional context data, which are usually used to preprocess and postprocess the call. This metadata processing is usually not reusable, and it is hard to test. **Listing 9** shows the delegation of functionality to a separate class.

Interceptors are executed in the scope of the target bean. They participate in the transaction, they are executed in the security context, and their lifecycle is identical to the intercepted bean. Reusable cross-cutting functionality can be easily factored out into an independent, reusable bean and injected into the interceptor. **Listing 10** shows a reusable audit class.

There are no specific requirements for the reusable class. It can be any POJO (as in **Listing 10**), a Contexts and Dependency Injection (CDI)–managed bean, or an EJB 3.1 bean. The current execution contexts (transactions, security, and context data) are passed to the injected class as well.

The interceptor method denoted with the @AroundInvoke annotation should implement only the generic code necessary to interpret InvocationContext. Any reusable aspects or functionality should be extracted into separate methods or independent classes. Such a separation simplifies unit testing of the reusable code, as well as integration testing for the interceptor implementation itself.

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

```
<beans>
 <interceptors>
    <class>(...).aop.interceptors.NicenessExtender</class>
    <class>(...).aop.interceptors.PersonalNicenessExtender</class>
  </interceptors>
</beans>
```

Download all listings in this issue as text

There Are Some Limits

Unlike AOP frameworks with "before" and "after" advice (that is, interception points), interceptors are only able to wrap the whole method with the @AroundInvoke annotation. Custom actions can be performed before and after the invocation, but you have to trigger the invocation of the target manually with the InvocationContext#proceed() call yourself.

Also, AOP frameworks come with powerful tools to find the interception points (*pointcuts*, in the AOP terminology). Unlike Java EE 6, AOP frameworks are able to decorate not only methods but also constructors and even fields. Java EE 6 interceptors are limited to method decoration only.

In AOP, you can use regular expressions, custom languages, and APIs to locate the pointcuts, whereas in Java EE 6 you are usually relying on annotations (@Interceptor, @InterceptorBinding, and @Stereotype) or XML deployment descriptors (ejb-jar.xml and beans.xml) to control the interception. **Listing 11** shows lifecycle callbacks in an interceptor.

However, you are not limited to intercepting only business methods. A lifecycle callback interceptor is also able to intercept object creation and destruction

(see **Listing 11**). Instead of using the @AroundInvoke annotation, @PostConstruct and @PreDestroy annotations are used to denote methods that are going to be invoked after the intercepted object is created and destroyed. In lifecycle callback methods, the InvocationContext parameter is optional.

Nothing Is Impossible

So far, we have covered the usual way of intercepting things. Java EE 6 comes with a Service Provider Interface (SPI), which allows you to intercept deployment time events and control the configuration of the system. During the application's boot time, CDI builds its metamodel from the types, annotations, and beans.xml file.

During the recognition, the container throws events that carry the meta-information. But the events are not read-only. You can enrich and even replace the metainformation entirely. You can add or remove annotations (such as @InterceptorBinding) and react to discovery or injection of beans. You can easily react to your custom annotations or add interceptors dependent on class names, the class locations, or whatever trigger you want.

A CDI extension is registered with the JDK's JAR extension mechanism. You have to put a file with the name of the extension (javax.enterprise.inject.spi.Extension) and the name of the implementation (your extension) as content. The "extension" file resides inside the WAR file in the WEB-INF/classes/META-INF/services/ folder. The Extension interface is just a marker—you are not forced to implement any methods. An extension listens to events and is able to change their state. **Listing 12** shows adding the HiPrefixer dynamically. The BootObserver annotation in **Listing 12** listens to all ProcessAnnotatedType events. The ProcessAnnotatedType event wraps an AnnotatedType instance that represents a Java class (usually a managed bean or EJB bean).

Unfortunately, AnnotatedType is an immutable, read-only class. To add additional annotations, you will have to build your own mutable AnnotatedType implementation, as shown in **Listing 13**.

To add a custom interceptor without changing the code at startup, an additional InterceptorBinding annotation, @Hi, has to be added at boot time. The list of already existing annotations at the UniversalGreeter bean has to be extended with @Hi. The field Set<Annotation> annotations in AnnotatedTypeWrapper (**Listing 13**) has exactly this purpose. The method getAnnotations() merges the custom annotations with annotations discovered by the CDI runtime.

An annotation instance cannot be just created. You have to use a helper class instead:

```
public class HiInstance implements
Hi{
  @Override
  public Class<? extends
Annotation> annotationType() {
     return Hi.class;
  }
 }
```

LISTING 12 LISTING 13 / LISTING 14 / LISTING 15

```
public class BootObserver implements Extension{
  void processUniversalGreeter(@Observes ProcessAnnotatedType event) {
    AnnotatedType annotatedType = event.getAnnotatedType();
    Class javaClass = annotatedType.getJavaClass();
    if(javaClass.getName().endsWith("UniversalGreeter")){
     AnnotatedTypeWrapper atw = new AnnotatedTypeWrapper(annotatedType);
     atw.addAnnotation(new HiInstance());
     event.setAnnotatedType(atw);
    }
  }
}
```

Download all listings in this issue as text

In the BootObserver class, we pass the HiInstance helper instead of the @Hi annotation.

Cleaner Code with Decorators

Our interceptor sample has a fundamental problem. Interceptors work well as generic decorators, but our sample relies on a specific method signature. A different method return type (for example, int) would result in a ClassCastException at runtime. Also, decoration of specific methods with custom cross-cutting functionality usually results in a lot of plumbing. **Listing 14** is an (ugly) implementation of method-specific behavior.

With the code in **Listing 14**, the interceptor would have to examine the method name or parameters to apply method-specific behavior. You could also apply method-specific interceptors on each method, but a type-safe implementation of the decorator pattern is far easier to maintain.

Method-dependent cross-cutting concerns can be cleanly implemented with @Decorator. The @Decorator annotation in CDI can be considered to be a type-safe interceptor. **Listing 15** shows

method-specific decoration with @Decorator.

By implementing the interface, you will have to implement all methods in @Decorator (see **Listing 15**). The application server is going to inject your decorator instead of the real class. To achieve that, you have to introduce an interface shared by both the decorator and the Greeter bean. Now the interface is implemented by both classes, GreetDecorator as well as UniversalGreeter:

```
public class UniversalGreeter
implements Greeter {}
```

The Greeter interface is injected into the GreetingController instead of the UniversalGreeter realization. As in the interceptor case, you also have to activate the decorators in beans.xml first:

```
<beans>
 <decorators>
   <class>(...).aop.decorators.
GreetDecorator</class>
 </decorators>
</beans>
```

By making the decorator abstract, you are not forced to implement all methods of the Greeter interface. **Listing 16** shows partial decoration with abstract decorators.

You can choose methods to decorate and ignore the rest. Multiple abstract decorators can be applied on a subset of methods to cover different aspects. Unlike interceptors, a decorator cannot be applied with an @Stereotype, because it is not sufficient for decoration. Decorators require an interface, which cannot be specified with the @Stereotype annotation.

LISTING 16

```
@Decorator
public abstract class GreetDecorator implements Greeter{
  @Inject @Delegate
  Greeter greeter;

  @Override
  public String greet() {
    return "Good " + greeter.greet();
  }
}
```

Download all listings in this issue as text

Back to the Real World

All questions related to injection, decoration, or interception in Java EE 6 can usually be answered with "yes." Therefore, you should refine such questions and ask whether it is possible without custom extensions.

Although it seems like everything is possible in Java EE, you need only a subset of the AOP capabilities introduced here in real-world projects. In fact, I rarely use @InterceptorBinding and mostly rely on the pure @Interceptors annotation. I also tend to intercept in an all-or-nothing fashion all methods of a class and introduce @Interceptors only if the plumbing can be provably reduced.

Because the vast majority of all common cross-cutting concerns come with the platform, rarely will you find an obvious use case for AOP and, thus, for interceptors or decorators in Java EE. `</article>`

LEARN MORE

- JSR 330 (Dependency Injection for Java)
- Seam Solder
- CODI
- CanDI
- *Real World Java EE Night Hacks—Dissecting the Business Tier* (press.adam-bien.com, 2011)

JULIEN PONGE

BIO

Part 2

Clustering and High Availability Made Simple with GlassFish

Enable high-availability support for applications and session failover in GlassFish clusters.

This article is Part 2 of a two-part series on the clustering features of GlassFish. The first article introduced centralized cluster provisioning and management of GlassFish servers. This article discusses enabling high availability for applications in GlassFish clusters.

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

In Part 1, I discussed two orthogonal problems that arise in production: providing high availability of applications and absorbing traffic load. The solution lies in scaling applications horizontally by running not just one server but a *cluster* of servers.

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

TaskEE: a Stateful Application

The ClockEE example application of the previous article was great for showing the centralized administration of a GlassFish cluster. That being said, all it did was show the current date, so it had no session state that would be useful for illustrating high-availability features. Therefore, let's introduce a new application called TaskEE (see **Figure 1**).

TaskEE is a simple task management Web application by which tasks can be added and removed. It does not employ any persistence, such as a relational database. To keep things simple and to illustrate state propagation, the task list of a user is determined solely by the user's Web session. Hence, opening two different Web browsers allows the management of two distinct task lists, even if the browsers are physically running on the same machine.

TaskEE is a very simple Web application. The session-scoped Contexts and Dependency Injection (CDI) bean shown in **Listing 1** defines the task list.

Thanks to CDI and the session scope that we applied, injection of instances of this class is transparently associated to the Servlet session context. We expose a single Servlet as a controller, delegating to CDI-injected, session-scoped instances of TaskList while using a single JavaServer Pages bean as the view, as shown in **Listing 2**.

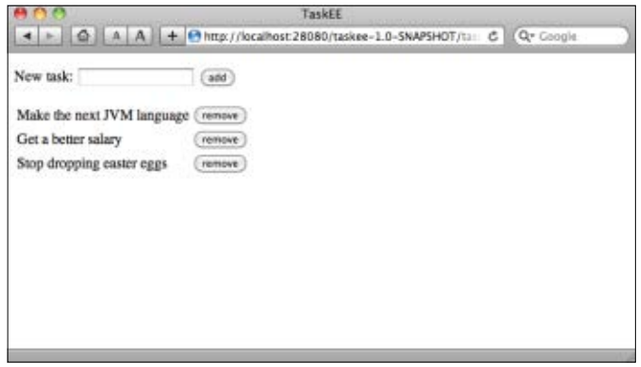


Figure 1

We left the remainder of the Servlet and JavaServer Pages code aside. The complete source code is available in the TaskEE folder.

Memory Session State Replication

By default, when an application is deployed to the instances of a cluster, runtime state is confined to each instance. Especially, state contained in Web sessions or stateful Enterprise JavaBeans (EJB) beans remains local. Having a cluster for high-availability and load-balancing purposes is useful if, and only if, an incoming request can be honored by any instance. The capability of having an incoming request be handled by any instance makes it possible to support failover scenarios when an instance fails or needs to be taken offline temporarily for maintenance. Hence, we need application session state to be replicated across the instances of a cluster so that transparent failover is possible.

GlassFish can replicate both HTTP session and stateful EJB beans state data to the instances

PHOTOGRAPH BY
MATT BOSTOCK/GETTY IMAGES

of a cluster. To do that, it relies on the Shoal group management service (GMS) that all instances refer to. This service is based on multicast User Datagram Protocol (UDP) socket communications for one-to-many communications and the Grizzly project for point-to-point communications. This service provides efficient propagation of state change events. It also absorbs instances of volatility in a group, and it provides the support for failover and recovery when instances leave. The GMS uses a consistent hash-based algorithm whereby instances know which instance hosts a replica of some state. This reduces the network multicast broadcast traffic. A simpler approach would be for an instance to broadcast a lookup to all instances in a group, as was the case in earlier versions of GlassFish.

To briefly illustrate how high availability works in GlassFish, let's consider the case of HTTP sessions. A session is obtained on the server side through the use of a cookie that is carried along with each request and response. Requests are always processed by the same instance for performance reasons (we will see later how to do that). Objects stored in an HTTP session are retrieved through the value of the cookie (JSESSIONID, in the case of GlassFish), which acts as a key. If the instance disappears, failover happens by processing the requests with another active instance in the cluster. Provided that cookies are properly managed by the front-end server and the cookie domain policies are correct, the new instance can access the same HTTP session data because it is available either locally in cache or from another instance.

ON DUTY
Making a Java EE application **ready for high availability** is mostly an infrastructure duty.

The technical requirements for high availability to work in GlassFish are the following:

- Instances must be on the same subnetwork; otherwise, multicast UDP communications of the GMS won't work.
- The web.xml descriptor of Web applications must have a <distributable/> element.
- Objects stored in HTTP sessions and stateful EJB beans must implement java.io.Serializable; otherwise, their state cannot be propagated to the other instances.
- Cluster instances and applications must have high availability enabled at deployment time. This makes it possible to disable state replication for some applications that do not need it, which reduces the GMS communications overhead.
- The physical hosts for instances should have their clocks synchronized as closely as possible to avoid inconsistencies.

It is very interesting to note that making a Java EE application ready for high availability is mostly an infrastructure duty. There is a requirement for replicated Java classes to be serializable and a requirement for having a specific XML element in the web.xml descriptor. There is no further requirement from the application code point of view.

LISTING 1 LISTING 2 / LISTING 3

```
@Named
@SessionScoped
public class TaskList implements Serializable {

  private List<String> tasks = new ArrayList<String>() {
    {
      add("Wash the dishes");
      add("Invent the next JVM language");
      add("Get a better salary");
    }
  };

  public List<String> getTasks() {
    return unmodifiableList(tasks);
  }

  public void add(String task) {
    tasks.add(task);
  }

  public void remove(String task) {
    tasks.remove(task);
  }
}
```

Download all listings in this issue as text

Getting the Cluster Ready for High Availability

There are further useful commands when dealing with a cluster. The command shown in **Listing 3** checks the health of a cluster by giving us useful information about the running and stopped instances.

Instances must be on the same network subsystem for state replication to work, because part of the communications are made using UDP multicast sockets. We can check that this requirement is satisfied by using the command shown in **Listing 4**.

You need to launch the same command from each host for the test to work. The –bindaddress parameter is optional and might be useful only with multiple network interfaces. You should see replies from all hosts that have a running instance as part of your cluster. If you cannot see some of your hosts, you need to investigate why. Indeed, it is very likely that the GMS communica-

tions will not work, resulting in no state replication at all.

The last preparation of our cluster is setting a property to globally enable high availability. You can tune this setting at the instance level or container level, for example, enabling it for the Web container but not for the EJB container. In most cases, you will be just fine enabling it at the cluster level so that all instances and all containers support high availability. The property name pattern is `<cluster name>-config.availability-service.availability-enabled`, and in our case, it is set as shown in **Listing 5**.

Making TaskEE Highly Available

At this stage, our cluster has been properly configured for providing high availability through state replication, so now is the time to deploy TaskEE, as shown in **Listing 6**.

The deployment command is the same as before except for the addition of the `--availabilityenabled` flag, which we set to `true`. As the name suggests, this flag triggers session replication and failover support in the DAS configuration, which then informs the GMS plus the EJB and Web containers.

A good smoke test is to connect to the HTTP connectors of each instance with the URL to the TaskEE application root context, as in **Figure 1**, and add and remove a few tasks. While instances running on the same host/node should have their sessions synchronized, pointing to another instance should yield a fresh session with a different task list. But . . . didn't we just trigger HTTP session replication? Of course we did! Sessions differing from one host to the other is not a bug, but instead it is a genuine feature.

Indeed, remember that HTTP sessions are stored on the server side through a key that is a cookie value exchanged back and forth with the HTTP client upon each request. A cookie is a critical piece of information that must not be exchanged with a server that is not controlled by the server's provider. Hence, cookies are restricted to a certain domain and subdomains. This is why sessions differ if you connect to TaskEE on, say, `localhost`, `stardust`, or `caramoustra` if each of those hosts is not pointing to the same system.

Adding a Load Balancer

Intuitively, users of our application will connect to a single host or URL, while the fact that a cluster is running the application must be transparent to them. To do that, a *load balancer* is used. Simply put, a load balancer is a front-end HTTP server that dispatches requests to the running instances and forwards their responses back to the HTTP clients. Because cookies are exchanged with a single host, HTTP session failover is possible since all instances share a uniform session data space. The same applies to stateful EJB beans, although their association with a given request is more often a result of the HTTP session itself.

There are various strategies for a load balancer to dispatch requests. Simple techniques include randomization and round-robin. More-elaborate techniques try to always dispatch a given HTTP client's requests to the same instance, changing only in failover scenarios to cater to the disappearance of the instance. This is also referred to as *sticky load balancing*. Sticky load balancing is useful because targeting the same instance maximizes locality of session data, while round-robin and randomized dispatching induce more replication work and network traffic in the cluster network.

Connecting a GlassFish server to a front-end HTTP server is very easy. Popular techniques include Apache HTTPD and a special module called `mod_jk` that uses a specific binary protocol called *AJP*. AJP is helpful because some tasks, such as serving static content, can be performed directly by Apache HTTPD, while the remainder of the work is performed by GlassFish instances.

Customers of the commercially supported Oracle GlassFish Server can obtain a dedicated load balancer plug-in, which features an easy-to-use IzPack-based installation to facilitate the connection with Apache HTTPD, Microsoft IIS, and Oracle iPlanet Web Server. It also provides an overall better and secured integration between a front-end HTTP server and a GlassFish cluster.

Users of the GlassFish Open Source Edition are encouraged to use a combination of Apache HTTPD and `mod_jk`. There are many alternative HTTP servers that are capable of performing load balancing as well. Because configuring `mod_jk` has been largely documented, we will instead use Lighttpd for this article. It is a solid lightweight HTTP server with a proven track record running popular internet Websites, and its simple and clean configuration file

LISTING 4 LISTING 5 / LISTING 6

```
$ asadmin validate-multicast --bindaddress 192.168.56.1
Will use port 2048
Will use address 228.9.3.1
Will use bind interface 192.168.56.1
Will use wait period 2,000 (in milliseconds)

Listening for data...
Sending message with content "infinity.home" every 2,000 milliseconds
Received data from infinity.home (loopback)
Received data from ubuntu-vm.home
Received data from infinity.home
Exiting after 20 seconds. To change this timeout, use the --timeout command line
option.
Command validate-multicast executed successfully.
```

Download all listings in this issue as text

syntax makes it very appealing here.

For testing purposes, the lighttpd.conf configuration file shown in **Listing 7** is sufficient to launch a load balancer on port 1981 to our three instances.

The configuration file is relatively self-explanatory. The hash load balancing strategy dispatches requests of a given client to the same instance for as long as the instance remains running. When Lighttpd detects that an instance cannot be reached while trying to dispatch a request, Lighttpd automatically selects another instance, inducing a session failover.

Based on the configuration file above, we can launch Lighttpd as a load balancer using the following command:

```
$ lighttpd -D -f lighttpd.conf
2011-07-01 15:23:59: (log.c.166)
server started
```

Failover in Action

We can connect to the application on port 1981 and reach the application, as illustrated in **Figure 2**. We see that some instance successfully handled our request.

Using a Web inspector found in popular Web browsers, we can get a hint regarding which instance was first assigned to us by the load balancer. The cookie value JREPLICA contains the name of the instance, which is instance2 in the example shown in **Figure 2**.

We can put the failover mechanism in action by shutting down instance2 and reloading the page or by performing an action, such as adding or deleting a task. Another of the two remaining instances processes the request without any loss of session information. You can further try the mechanism by starting and stopping instances at will to see the effects.

Troubleshooting note: You might encounter certain problems with sessions not working as they should in the case of failover on an instance running on a different host than the previous one. This is most likely due to your particular combination of network, load balancer, and GlassFish server settings causing cookies to be issued for different domains. It is possible to explicitly instruct a GlassFish server to define the cookie domain value that you expect based on the HTTP load balancer public URL. To do that, you need to add a WEB-INF/glassfish-web.xml file in your Web applications with content similar to the code shown in **Listing 8** for connections to http://my.wonderfulapp.com/.

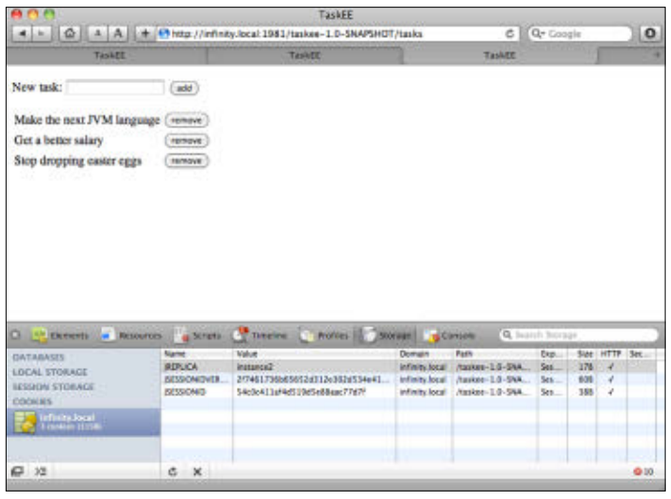


Figure 2

LISTING 7 LISTING 8

```
server.document-root = "."
server.port = 1981
server.modules += ( "mod_proxy" )
proxy.balance = "hash"
proxy.server = ( "" =>
  (
    (
      "host" => "127.0.0.1",
      "port" => 28080
    ),
    (
      "host" => "127.0.0.1",
      "port" => 28081
    ),
    (
      "host" => "192.168.56.101",
      "port" => 28080
    )
  )
)
```

Download all listings in this issue as text

Conclusion

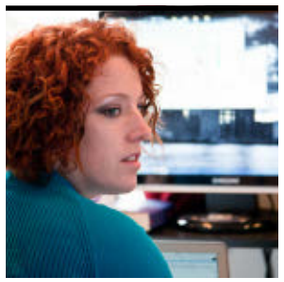
This article concludes the GlassFish clustering series. The first article focused on the centralized provisioning and management of a cluster; this one dealt with enabling high-availability support for applications and session failover.

We also discussed the options for setting up a front-end HTTP load balancer for a GlassFish cluster, choosing a less-documented yet easy-to-test option without loss of generality. While we did not illustrate stateful EJB session replication in our examples, all points remain valid.

GlassFish is not just a convenient server in development contexts; it is also a full-fledged server for production when horizontal scaling is required both for performance and high-availability purposes. It provides an elegant and easy-to-grasp tool for centrally provisioning and administering cluster instances. `</article>`

LEARN MORE

- "GlassFish Server Load Balancing Plug-in Installation and Setup"
- *Beginning Java EE 6 with GlassFish 3* by Antonio Goncalves (Apress, 2010)
- *The Java EE 6 Tutorial: Basic Concepts*, fourth edition (Prentice Hall, 2010)



TRISHA GEE

BIO

Sharing Data Among Threads Without Contention

Use the Disruptor framework to do the heavy lifting for concurrent programming.

The London Multi-Asset Exchange (LMAX) Disruptor is an open source concurrency framework that recently won the 2011 Duke's Choice Award for Innovative Programming Framework. In this article, I use diagrams to describe what the Disruptor is; what it does; and, to some extent, how it works.

What Is the Disruptor?

The Disruptor is a framework for interthread communication (ITC), that is, the sharing of data among threads. LMAX created the Disruptor as part of its reliable messaging architecture and developed it into an extremely fast way of handing off data between different components.

Using *mechanical sympathy* (an understanding of how the underlying hardware works), fundamental computer science, and domain-driven design, the Disruptor has evolved into a framework that developers can use to do much of the heavy lifting for concurrent programming.

In many architectures, it is common to use a queue to share data (that is, pass messages) among threads. **Figure 1** shows an example of passing messages between stages using a queue. (Each little blue spinner is meant to represent a thread.)

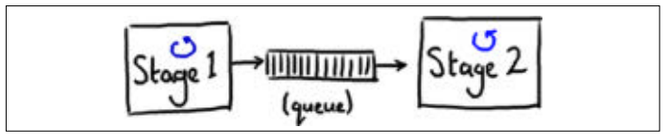


Figure 1

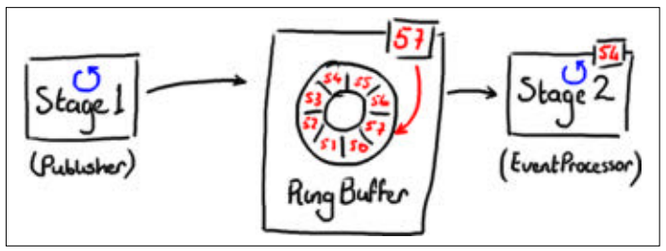


Figure 2

This architecture allows the producing thread (Stage 1 in **Figure 1**) to move on to the next piece of work if Stage 2 is too busy to accept the data immediately, which provides a way to deal with bursts of traffic in a system. The queue acts as a buffer between the threads.

The Disruptor is a 2011 Duke's Choice Award winner.

In its simplest form, the Disruptor can be used in place of the queue in the architecture shown in **Figure 2**, in which messages are passed between stages using the Disruptor.

The data structure storing the messages is a RingBuffer, implemented as an array. Stage 1 places items into the RingBuffer, and Stage 2 reads items from the RingBuffer.

You'll see in **Figure 2** that each spot in the RingBuffer is indexed by a sequence number, and the RingBuffer tracks the highest (most recent) sequence number, which is the index pointing to the last item in the RingBuffer. This sequence number continually increases as more data is added to the ring.

The key thing about the Disruptor is that it was designed to have zero contention within the framework. This is achieved by following the single-writer principle—only one thing can ever write to a single piece of data. Following this principle eliminates the need for expensive lock or compare and swap (CAS) operations, which is one of the reasons the Disruptor is so fast.

One source of contention has been removed by having the RingBuffer and each EventProcessor track their own sequence numbers. This way, the only thing that ever updates a sequence number is the thing that owns the sequence number. This concept is covered in more detail when I describe writing to and reading from the RingBuffer.

PHOTOGRAPH BY JOHN BLYTHE

Publishing to the Disruptor

Writing to the RingBuffer is achieved using a two-phase commit. First, Stage 1 (the Publisher) has to determine the next available slot in the RingBuffer, as shown in **Figure 3**.

The RingBuffer has the sequence number of the most recently written-to slot (Slot 18 in **Figure 3**) and, therefore, it can determine the next sequence number and, consequently, the corresponding slot in the array.

The RingBuffer determines whether the next slot is free by inspecting the sequence number of any EventProcessor that is reading from the RingBuffer.

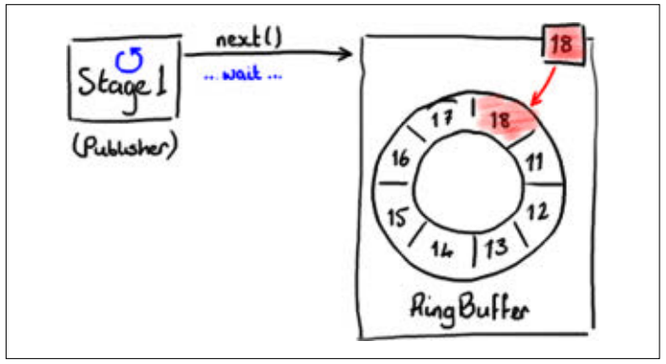


Figure 3

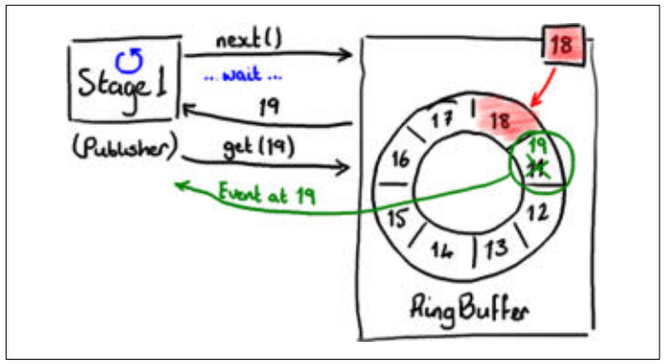


Figure 4

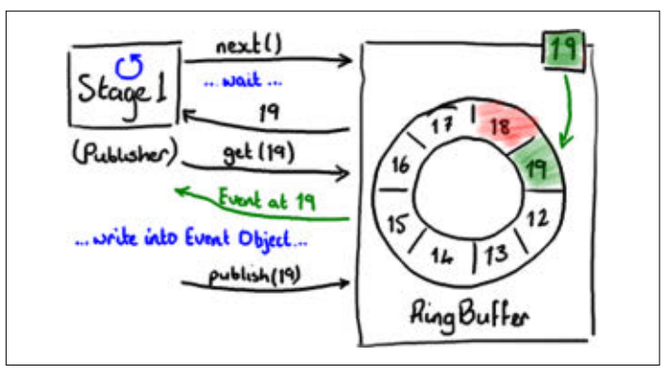


Figure 5

Figure 4 shows the next slot being claimed.

When the Publisher gets the next sequence number, it asks the RingBuffer for the object in that slot. The Publisher can then do what it likes to that object. You can think of the slot as a simple container that you write values into.

All the while, the RingBuffer sequence number is still Slot 18, so anything reading from the RingBuffer will not be able to read the event in Slot 19 while the Publisher is still working on it.

Figure 5 shows changes being committed to the RingBuffer.

Finally, when the Publisher is finished writing things into Slot 19, it tells the RingBuffer to publish the item that is in Slot 19. At this point, the RingBuffer updates its sequence number, and anything that wants to read from the RingBuffer can see the item in Slot 19.

Reading from the RingBuffer

The Disruptor framework includes a BatchEventProcessor that will read events from the RingBuffer for you, but I'm going to outline how that works to highlight the design.

While the Publisher needed to ask the RingBuffer for the number of the next slot to write to, an EventProcessor, which is kind of like a consumer except it doesn't actually consume (remove) things from the RingBuffer, will track the last sequence number it processed and ask for the next sequence number it wants.

Figure 6 shows the EventProcessor waiting for the next expected sequence number.

Rather than asking the RingBuffer directly for its sequence number, an EventProcessor has a SequenceBarrier do this job. This detail is not important for the case we're looking at now, but its purpose will become apparent later.

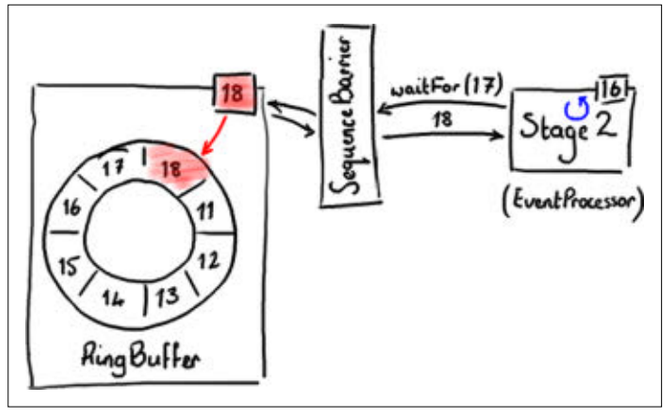


Figure 6

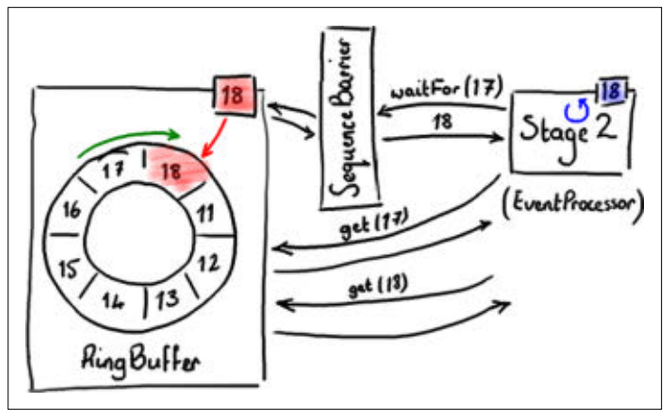


Figure 7

In **Figure 6**, Stage 2, the EventProcessor has seen up to sequence number 16. It wants the item at Slot 17 next, so it calls waitFor(17) on the SequenceBarrier. Stage 2 can happily hang around waiting for the next sequence number, because it might be that nothing else has written to the RingBuffer. If so, there's nothing else to process. However, in the case shown in **Figure 6**, the RingBuffer has been populated up to Slot 18, so waitFor returns the number 18, telling the EventProcessor it can safely read anything up to and including Slot 18, as shown in **Figure 7**.

This methodology provides some really nice batching behavior, which you can see in the BatchEventProcessor code. The code simply asks the RingBuffer for every item from the next value it needs up to the highest available sequence number.

You can use this batching code by implementing EventHandler. There are examples of how to use batching in the Disruptor performance tests, for example FizzBuzzEventHandler.

Is It a Low-Latency Queue?

Sure, it can be used as such. We have figures from testing early versions of the Disruptor that show how much faster it is than ArrayBlockingQueue for a three-

stage pipeline running on a 2.2 GHz Intel Core i7-2720QM processor using Java 1.6.0_25 64-bit on Ubuntu 11.04. Table 1 shows the latency per hop in the pipeline. For more details about this test, see the Disruptor technical paper.

But don't think that these latency figures mean the Disruptor is a specialist solution for a specific performance niche, because it's not.

More Cool Stuff

One of the interesting things about the Disruptor is how it supports graphs of dependencies between system components without incurring any of the cost of contention usually associated with sharing data among threads.

The Disruptor achieves contention-free design by adhering to the single-writer principle, so each bit of data can be written only by a single thread. However, that doesn't mean you can't have *multiple readers*. And that's exactly what the Disruptor enables.

The system the Disruptor was originally designed to support had a staged pipeline of events that needed to happen in a particular order, which is not an uncommon requirement in an enterprise application. **Figure 8** shows a standard three-stage pipeline.

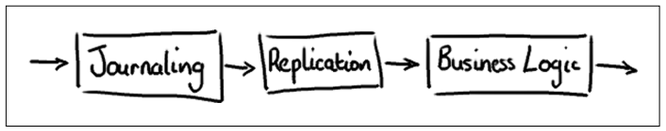


Figure 8

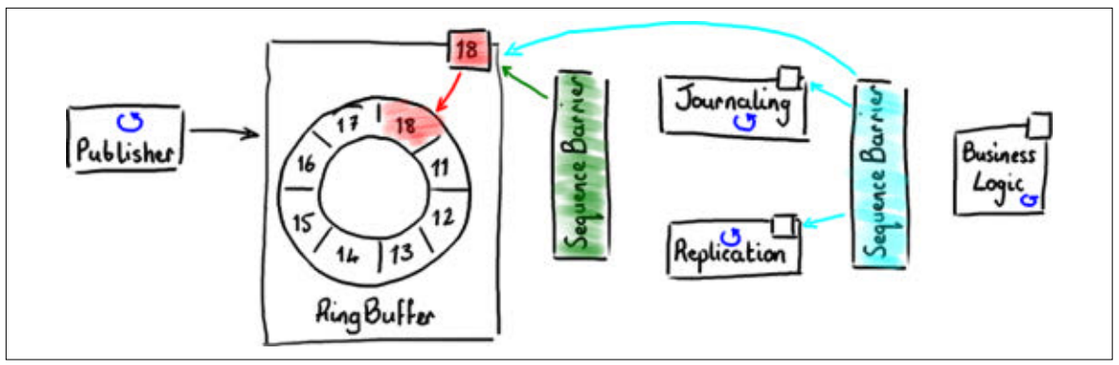


Figure 9

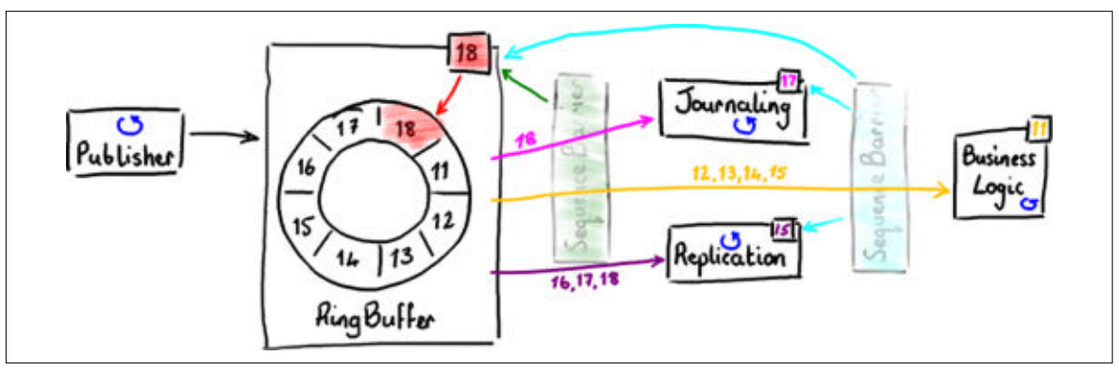


Figure 10

First, each event is written to disk (journaling) for recovery purposes. Next, the events are replicated to secondary servers. Only after both these stages occur can the system be allowed to proceed with the real business logic.

Doing these tasks sequentially is a logical approach, but it is not the most efficient. Journaling and replication can happen in parallel, because they are not dependent on each other. But the business logic is dependent upon the other two events having taken place. **Figure 9** shows how to parallelize the dependencies.

If you wire up the dependencies using the Disruptor, the first two stages (journaling and replication) can read directly from the RingBuffer. As with the simple case in **Figure 7**, they both use a single sequence barrier to get the next available sequence number from the RingBuffer. They track their own sequence numbers, so they know which events they've seen, and can process batches of events using a BatchEventProcessor.

The business logic can also read events from the same RingBuffer, but it is limited to processing only those events that the first two stages have successfully dealt with. This is achieved with a second SequenceBarrier, which is configured to inspect the sequence numbers of the journaling EventProcessor and the replication EventProcessor and return the lower of the two numbers when it is asked for a number that is safe to read up to.

After every EventProcessor has used the sequence barriers to determine the events that are safe to read from the RingBuffer, it can request them from the RingBuffer (see **Figure 10**).

There's a lot of reading of different sequence numbers—those from the RingBuffer, the journaling EventProcessor, and the replication EventProcessor—but only one thing can ever increment each sequence number. This ensures no contention on these values among the different threads.

What About Multiple Publishers?

The Disruptor also supports multiple publishers writing to the RingBuffer. However, because that entails two different things writing to the same place, this is a scenario in which you will get contention. The Disruptor provides different ClaimStrategy types for cases where you have more than one publisher.

Conclusion

I've talked at a high level about how the Disruptor is a framework for sharing data among threads in a very high-performance fashion. I've also described a little about how it works. I didn't touch on features such as more-advanced event processors and various strategies for claiming a slot in the RingBuffer and waiting for the next sequence in the RingBuffer. The Disruptor is open source—go explore the code! `</article>`

LEARN MORE

- Disruptor downloads
- Disruptor blogs and articles

//fix this /

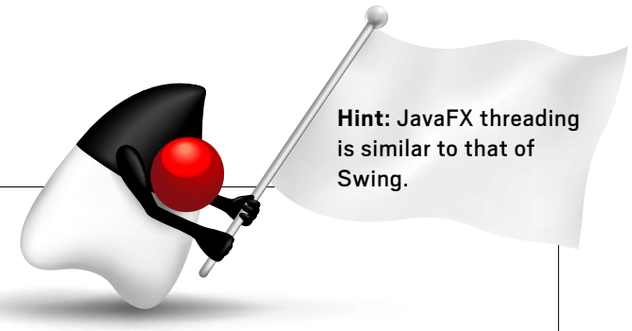
In the last issue, Marek Piechut created a Swing window that displayed a simple table and challenged readers to find the error in his code, because the resulting table was not what he wanted.

The correct answer is #3: you cannot simply reuse one instance of Component in Container. Container adds method calls (addImpl), removing the previous parent from the added component (see Javadoc for the Component.addImpl method). This way it will be rendered only in the last place it's added.

This issue's challenge comes from Jonathan Giles, a software engineer in the JavaFX team at Oracle.

1 THE PROBLEM

JavaFX 2.0 allows for rich client applications to be built in pure Java or in any other Java Virtual Machine language. (Excellent support already exists in languages such as Groovy and Scala—see GroovyFX and ScalaFX.) JavaFX has a modern, high-performance graphics engine sitting beneath a powerful scene graph with high-quality UI controls, as well as a number of other features. As with any UI toolkit, there are things to watch out for—here's one of them.



2 THE CODE

Consider the following Java code that demonstrates a thread retrieving data from a remote (and possibly slow) datasource, and updates a JavaFX ProgressBar control. Note that the isComplete() and getRemoteProgress() methods are not APIs—they are just demonstrative.

```
final ProgressBar pb = new ProgressBar(0);
new Thread(new Runnable() {
    public void run() {
        while (! isComplete()) {
            // Retrieve chunk of data...

            // Update progress value
            pb.setProgress(getRemoteProgress());
        }
    }
}).start();
```

3 WHAT'S THE FIX?

Is this code guaranteed to work in all environments? If not, what needs to change?

1) Yes. The ProgressBar will always correctly reflect the remote progress value.
2) Yes, but only if the ProgressBar is instantiated and added to the JavaFX scene from within the external thread.
3) No. The ProgressBar must always be updated from the JavaFX application thread.
4) No. JavaFX does not support working with the Thread class—everything must be done on a single JavaFX application thread.

GOT THE ANSWER?

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

ART BY I-HUA CHEN

EDITORIAL

Editor in Chief
Justin Kestelyn

Senior Managing Editor
Caroline Kvitka

Community Editors
Cassandra Clark, Sonya Barry, Yolande Poirier

Java in Action Editor
Michelle Kovac

Technology Editors
Janice Heiss, Tori Wieldt

Contributing Writer
Kevin Farnham

Contributing Editors
Blair Campbell, Claire Breen, Karen Perkins

DESIGN

Senior Creative Director
Francisco G Delgadillo

Design Director
Richard Merchán

Contributing Designers
Jaime Ferrand, Paulina McFarland, Chris Strach

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