// MAY/JUNE 2012 /

Java™ magazine

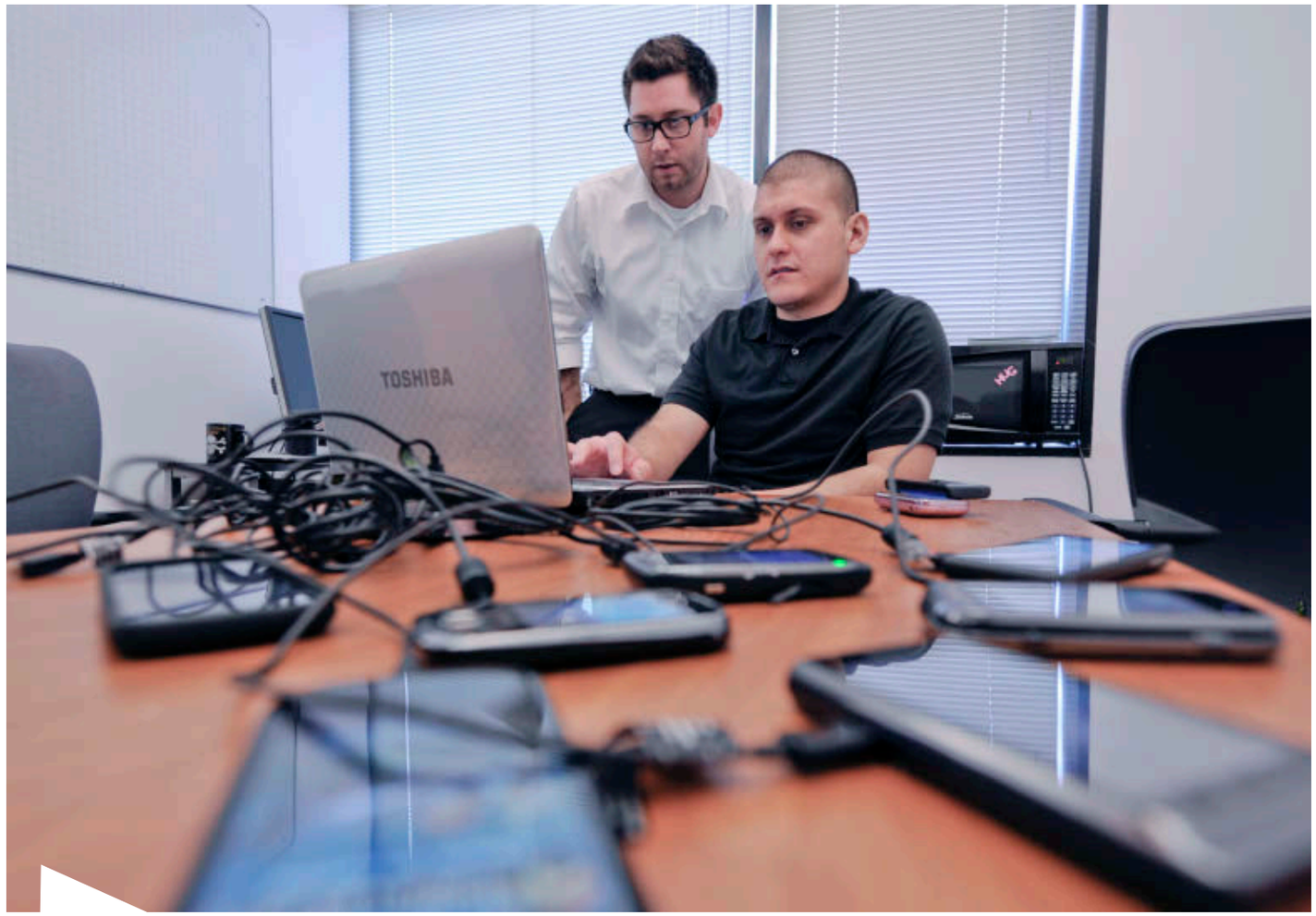
By and for the Java community

DEVICE CONVERGENCE

Jargon Technologies creates unique interactive and immersive experiences with Java 12

//table of contents /

COVER ART BY I-HUA CHEN
PHOTOGRAPH BY PHIL SALTONSTALL



What's not to love about crowdsourcing? Without a clear and obvious topic in mind for this note—and cursed with a follow-on case of writer's block—I went to you, the community, via Twitter for suggestions. There were several of them, but the desire for a status update about JDK 8 seemed to be a common theme. So, let's do that.

As previously announced, the GA date for JDK 8 is currently expected to be September 2013. The main drivers are, of course, the Lambda and Jigsaw projects. Each of these projects could arguably represent one of the most significant and anticipated changes ever to the Java platform, to say nothing of two of them.

The curiosity about JDK status offers a welcome opportunity for us to deep-dive into the OpenJDK project overall—for example, to explore its expanding role in the community ecosystem and its relationship to the Oracle JDK. In this issue, we've banked on that opportunity via an interview with Donald Smith, a member of Oracle's Java SE team who helps steward the OpenJDK community. As you'll learn, one of the project's main objectives—and great successes thus far—is to bring a diversity of viewpoints into the development process.

And speaking of diverse viewpoints: before you turn the page, I want to thank you for helping *Java Magazine* break the 100,000-subscriber barrier after only a handful of issues. Without your interest and support, this publication would not, and could not, exist. Do you see a pattern here?

Justin Kestelyn, Editor in Chief BIO

P.S. Starting in this issue, the "Java Champion Duke" next to a byline signifies the author's membership in that august community. They deserve it!

//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

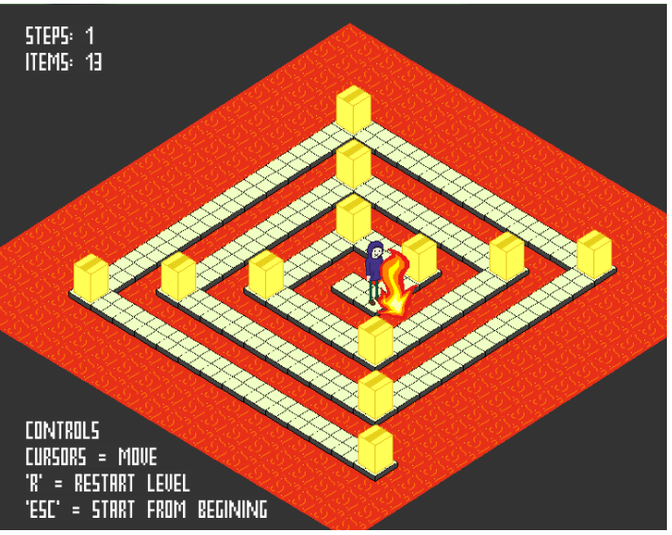
JAVA.NET ENHANCES ITS PLATFORM FOR JUGS

In recent months, **Michael Hüttermann** and others in the Java.net Java user groups (JUGs) community have devoted considerable time to updating and enhancing the platform and services Java.net offers to JUG members. New and updated features include the JUG events calendar and JUG Member Spotlights. Hüttermann focused special attention on JUG resources, a set of tools that allow you to browse the Java.net JUG community, submit your own JUG for membership, join mailing lists, and more. In addition, the resources include links to information on how to start/improve your JUG, blog postings, the JUG wiki, and the Adopt-a-JSR program.

One of the most popular updated features on the Java.net JUGs site is the Jug Profile Map, a Google Map that shows the location of JUGs across the world. More than 175 JUGs are already on the map. If you'd like to get your JUG onto the Java.net Jug Profile Map, just follow the instructions.

Global JUGs Growth Continues

Three new Java user groups (JUGs) have recently joined the growing Java.net JUG community. In **Croatia**, **Branko Mihaljević**, **Stjepan Matijašević**, **Hrvoje Đurđević**, **Marin Orlić**, and **Slavko Žnidarić** founded HUJAK, the Croatian Java User Association; the group also hosts a Croatian language home site, hujak.hr. In May, HUJAK will be collaborating with the Croatian Association of Oracle Users to hold Croatia's first ever "Java Day" event. In **Vilnius, Lithuania**, **Vaidas Pilkauskas**, **Kasparas Rudokas**, and **Aleksej Šipulia** founded the Vilnius Java User Group; the Vilnius JUG has also created a Google Group, and they are using Eventbrite to organize meetings, the first of which took place in March. Finally, **Mahmut Bulut** has worked with other Java professionals and enthusiasts in the **Eskişehir Province region of Turkey** to form the Eskişehir Java User Group. Congratulations and good luck to all of these new JUGs.



In this gallery at java_gaming.org, tutorial users can share the games that they have created.

DEVELOP YOUR OWN GAME

Swedish video gaming company Mojang is encouraging gamers to go beyond just playing its games by learning game programming. Jens Bergensten, lead designer and developer for Mojang's Minecraft game, created a getting-started guide to programming for beginners using the source code from "Catacomb Snatch" and the Eclipse integrated development environment. "Catacomb Snatch" is a popular 2-D shooting strategy game. The game was part of a Humble Bundle Mojam, a 60-hour charity programming marathon from which half a million dollars in proceeds went to charity. In addition to playing with "Catacomb Snatch" source code, gamers have access to Java 2-D and 3-D gaming resources. They can also share their games and enter competitions such as the one-hour "Let's Get Petite" competition.

JUG-Africa Raises Money for Congo

Following the March 2012 death of 200 Congolese citizens due to an explosion in Brazzaville, Congo, **Lamine Ba**, the leader of the Java user group (JUG) in Senegal, used Java to create an online payment Website to accept donations for the victims. Congo and Senegal are 2 of the 18 countries that are affiliated with JUG-Africa. This umbrella organization was created by **Max Bonbhel** in 2010 to share Java expertise among professional developers. For example, the community organized a dozen Java 7 JUG meetings in the summer of 2011. Ba hopes that JUG-Africa will be the network that will change the continent. "If we can create an impact through this project, then we can become a major force when it comes to supporting the too-frequent crises in Africa," he says.

MAX BONBHEL PHOTOGRAPH BY ALLEN MCINNIS/GETTY IMAGES

JAVA USER GROUP PROFILE

JUG Chennai Tamil Nadu, India

IN MOST INDIAN FAMILIES, JAVA IS VIEWED EXCLUSIVELY AS A MEANS FOR GETTING A GOOD JOB, says **Rajmahendra R Hegde**, a Java/Java EE project lead at Logica in Chennai, India. In 2006, though, Hegde discovered Java user groups (JUGs) on Java.net, and his interest in Java's community aspect was piqued. Hegde went on to found JUG Chennai in September 2010. Judging by its rapid growth (430 members today), Java as community is really catching on in India.

But it was by no means an easy start for JUG Chennai. Hegde reports that no more than seven people showed up for the group's first three meetings in 2010. But, the January 2011 meeting, in which he gave a talk about Java Virtual Machine languages, drew close to 20 people. That meeting was a major turning point in the Chennai Java community.

"For me, being a JUG lead is a tough and interesting job," Hegde says. "I am playing many roles that make me really happy: community leader, mentor, speaker, advisor to college students, and more—which I never did before as a normal Java professional."

JUG Chennai currently has at least one meeting per month, but the demand from the Chennai Java community is such that the group will soon move to two meetings per month. The group also hosts special events, such as the Chennai Java Summit; the Java 7 launch; and NetBeans certification, JavaFX, and Web 2.0 events.

JUG Chennai is also participating in JDuchess, a Java community for women, helping to form JDuchess Chennai. The group has also started a student awareness program through which it brings Java to students and involves them in Java community activities. In addition, JUG Chennai joined the Java Community Process and is active in the Adopt-a-JSR program. Several special interest groups have also formed.

Scenes from JUG Chennai, left to right: a speaker captivates the group, holding a Java Summit, celebrating the launch of Java 7

PHOTOGRAPHS COURTESY OF JUG CHENNAI

CALL FOR NOMINATIONS

Submit your entry for the Duke's Choice Awards 2012 awards, which celebrate extreme innovation in the world of Java technology. These awards are granted to the best and most-innovative projects using the Java platform. The primary judging criterion for this prestigious award is innovation, putting small developer shops on an equal footing with multinational giants. The 10 most innovative submissions will be chosen and showcased at Java One, September 29–October 4. There are no predetermined categories—it's all about innovation. Past winners are welcome to apply. Nominations are due July 20, 2012. Get your nomination form here.

Participants in Round Two of the Java Olympics showed laser focus during the competition, and then celebrated their achievements.

JIM WEAVER PHOTOGRAPH BY STEVE GRUBMAN

ROUND TWO OF JAVA OLYMPICS ENDS; FINALS LOOM

Round Two of the 2011–2012 Java Olympics finished in March, with Round One's top achievers (from 151 universities in Belarus, Kazakhstan, Russia, Ukraine, and Uzbekistan) competing in 11 cities across the region. Each competitor was provided with a PC preloaded with Oracle JDK 1.7u2 and NetBeans 7.1 and was presented with six problems (designed by a team of 12 Russian engineers) to solve using Java. **Alexander Belokrylov** of Oracle St. Petersburg reports that although all six problems were successfully solved, no one individual competitor solved all the problems.

Participants submitted their solutions to a remote server, where a tweaked version of the Ejudge Contest Management System analyzed the solutions. The results were delivered in real time as the competition progressed. "I followed all the contests," says Belokrylov. "It is a really exciting experience, like a horse race or football game!"

Round Two winners will travel to Astana, Kazakhstan in late April to compete in Round Three, the 2011–2012 Java Olympics championship finals.

Weaver and Chin Join Oracle

Jim Weaver

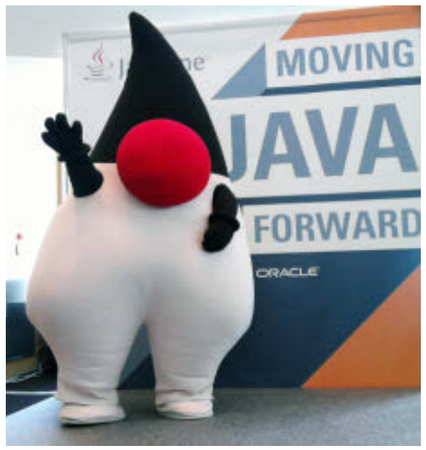
Stephen Chin

Jim Weaver and **Stephen Chin**—two recognized JavaFX technology luminaries and "rock star" speakers from the Java community—have joined Oracle's Java evangelist team. Jim Weaver is a Java and JavaFX developer, author, and speaker with a passion for helping rich-client Java and JavaFX become preferred technologies for new application development. He is the author of *Inside Java* (New Riders Pub, 1997) and *Beginning J2EE* (Wrox Press, 2003). Stephen Chin is a Java technical expert specializing in UI technologies, an open source hacker, and a seasoned conference speaker. Both Weaver and Chin are coauthors of *Pro JavaFX Platform* (Apress, 2009), the leading technical reference for JavaFX, and *Pro JavaFX 2* (Apress, 2012). This is a win for the entire Java community.

JavaSpotlight Podcast

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

MOVING JAVA FORWARD IN JAPAN



Left to right: Oracle's Nandini Ramani; Oracle's Jasper Potts; Duke welcoming attendees; Fujio Maruyama, visiting professor at Waseda University, explaining why Java is back

JavaOne Tokyo 2012 was held in Tokyo, Japan, April 4–6, 2012. With more than 1,100 attendees, this second of four regionally executed JavaOne conferences included informative keynote and technical sessions, vendor participation, an Oracle Technology Network lounge, and interactive local Java community involvement.

A major focus of the conference was the Java roadmap. At the Java Strategy keynote on the first day of JavaOne Tokyo, Oracle's Java engineering executives **Cameron Purdy**, **Nandini Ramani**, and **Henrik Stahl** stepped through Oracle's technology roadmap for Java SE, JavaFX, Java ME/embedded Java, and Java EE. The key takeaways: Java SE 8 will include Project Jigsaw and Project Lambda; Java ME/embedded Java will synchronize CLDC and JDK releases; and Java EE 7 will support elasticity for capacity on demand and multitenancy. **Sharat Chander**, group director of Java technology outreach at Oracle, also called on local Japanese developers to continue their active participation in the Java community by joining Oracle Technology Network.

PHOTOGRAPHS COURTESY OF ORACLE JAPAN

JAVA CHAMPION PROFILE

MICHAEL HÜTTERMANN: Up Close and Personal

*Java Champion **Michael Hüttermann**, developer, architect, consultant, and author of three books on agile development, recently took time off from his demanding schedule to tell* Java Magazine *a bit about his nonworking life.*

Java Magazine: Where did you grow up?

Hüttermann: I grew up in Ruhr, an urban area in North Rhine-Westphalia, Germany. Now I live in Cologne, the fourth-biggest city in Germany and a great place to live.

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

Hüttermann: My first computer was a Commodore 64. I played around and programmed with it in the early '80s. My very first languages included BASIC, Pascal, and, later, Cobol and Java. But the C64 was definitely the initial kickoff for me for computers and programming. I guess I share this history with many others.

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

Hüttermann: I'm a team player, and I've always very much enjoyed team sports. I've played soccer for years actively. But lately, I've run out of time to do so. I'm also a passive sportsman: I enjoy watching soccer matches on TV, and live matches in arenas as well.

Java Magazine: What else do you do in your spare time?

Hüttermann: Work! Also, I have a cat that is always a pleasure to play with. His name is Doctor Snuggles, named after the main character in a British/German/Dutch TV series that aired from 1979 to 1980. Watching the Doctor is a lot of fun; actually, it's very clear every day that cats are related to tigers. Cats are small tigers!

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

Hüttermann: Meeting many people and seeing new places. I don't regret being a freelancer today; after my first book was published, it was a good decision to leave my job then. I have much more freedom now and enjoy every minute. It's most important for me to be able to learn from others, and to find collaborative working environments.

Java Magazine: Anything else you want to share? Any plans?

Hüttermann: What makes me excited now is that my fourth book, on agile development, is in progress. I like writing; without writing, staying in hotels around the world every night would be pretty boring. And I hope that some people will find the book useful to read.

And so, the Java Champion goes back to his work. You can read more about Michael Hüttermann in the recent article "Agile ALM: A Conversation with Java Champion and ALM Expert Michael Hüttermann." You can also visit his blog and find him on Twitter (@huettermann).

EVENTS

OSCON *JULY 16–20, PORTLAND,OREGON*

Join the world's open source pioneers, builders, and innovators for five days of sessions and keynotes on open source languages, platforms, and development. OSCON features more than 200 speakers, hundreds of technologies, and 3,000 hackers. Tracks include Java and JVM Languages, Geek Lifestyle, JavaScript and HTML5, Mobile, Tools and Technologies, and much more.

PHOTOGRAPH BY JOE SOHM/GETTY IMAGES

MAY

The No Fluff Just Stuff Software Symposium Tour
Since 2001, the No Fluff Just Stuff (NFJS) Software Symposium Tour has delivered more than 275 events with more than 40,000 attendees. NFJS is known for knowledgeable speakers and timely presentations that cover the latest trends within the Java ecosystem and agility space. Upcoming symposium dates include the following:

Lone Star Software Symposium: Dallas
MAY 18–20
DALLAS, TEXAS

Central Ohio Software Symposium
JUNE 8–10
COLUMBUS, OHIO

Salt Lake Software Symposium
JUNE 29–30
SALT LAKE CITY, UTAH

Greater Maryland Software Symposium
JULY 13–14
COLUMBIA, MARYLAND

GeeCON 2012
MAY 16–18
POZNAŃ, POLAND
GeeCON focuses on Java- and Java Virtual Machine–based technologies, with special attention paid to dynamic languages such as Groovy and Ruby.

Maker Faire Bay Area
MAY 19–20
SAN MATEO, CALIFORNIA
See how Java will help build the future at this festival of invention, creativity, and resourcefulness.

GOTO Conferences
MAY 21–25
COPENHAGEN, DENMARK
MAY 24–26
AMSTERDAM, NETHERLANDS
These conferences are designed for software developers, IT architects, and project managers in the Java, mobile, .Net, open source, Lean/agile, architecture, new languages, and process communities.

JUNE

ÜberConf
JUNE 19–22
DENVER, COLORADO
Explore the evolving ecosystem of the Java platform in 135 technically focused sessions, including 25 hands-on workshops, centered around architecture, cloud, security, enterprise Java, languages on the Java Virtual Machine, build/test, mobility, and agility. ÜberConf is part of the No Fluff Just Stuff Software Symposium Tour.

JAZOON'12
JUNE 26–28
ZURICH, SWITZERLAND
Jazoon, the international conference on the modern art of software, will cover a broad range of topics and cater to a wide audience of professionals involved in software development. The main track covers programming and markup languages; frameworks, tools, platforms, and methods; and more.

JULY

The Developers Conference 2012
JULY 4–8
SÃO PAULO, BRAZIL
One of Brazil's largest developer conferences, this conference offers 35 tracks, several of which are Java related.

Java Forum
JULY 5
STUTTGART, GERMANY
The Java Forum offers participants the opportunity to inform themselves about topics related to Java and the Java environment. The forum includes basic lectures, presentations, and information about and demos of specific products.

FISL
JULY 25–28
PORT ALEGRE, BRAZIL
Sponsored by the Fórum Internacional do Software Livre (FISL), this event is one of the world's largest free software events, and includes technical, political, and social discussions about open source software.

JVM Language Summit
JULY 30–AUGUST 1
SANTA CLARA, CALIFORNIA
The 2012 JVM Language Summit is an open technical collaboration between language designers, compiler writers, tool builders, runtime engineers, and virtual machine architects.

DEVELOPERS WANT JSR 310 IN JAVA 8

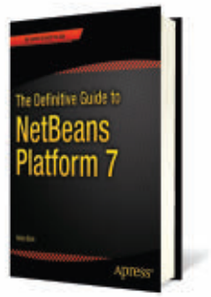
Should JSR 310 (Date and Time API) be included in Java 8? Based on a recent Java.net poll and the attention **Stephen Colebourne**'s JDK Enhancement Proposal (JEP) 150 has garnered, the broader Java developer community has a clear view on this: it's very important that JSR 310 is included in Java 8.

The Java.net poll asked, "How critical is it for JSR 310 (new Date and Time API) to be implemented in Java 8?" Rarely does a Java.net poll produce as unified a response as this one did. A full 75 percent of the 891 developers who voted consider including JSR 310 in Java 8 "very" important, agreeing with Colebourne's statement in JEP 150 that "the existing Java date and time classes are poor, mutable, and have unpredictable performance."

Java.net user heathm undoubtedly spoke for many in commenting, "Given the fact that we were expecting this in Java 7, I think it's something that's long overdue and must be included in Java 8." A late February update to JSR 310 states, "With successful progress it is intended that this JSR can be considered for inclusion in Java SE 8." Java developers have made it abundantly clear that they prefer that outcome.

JAVA BOOKS

EXPERT ORACLE AND JAVA SECURITY: PROGRAMMING SECURE ORACLE DATABASE APPLICATIONS WITH JAVA
By David Coffin
Apress (September 2011)
This book provides resources that every Java and Oracle database application programmer needs to ensure that you have guarded the security of the data and identities entrusted to you. You'll learn to consider potential vulnerabilities and to apply best practices in secure Java and PL/SQL coding. Author David Coffin shows you how to develop code to encrypt data in transit and at rest; to accomplish single sign-on with Oracle proxy connections; and to generate and distribute two-factor authentication tokens from the Oracle server using pagers, cell phones, and e-mail.



THE DEFINITIVE GUIDE TO NETBEANS PLATFORM 7
By Heiko Böck
Apress (December 2011)
The Definitive Guide to NetBeans Platform 7 is a thorough and authoritative introduction to the NetBeans platform. The book provides a completely updated definitive guide to the NetBeans platform, using the latest APIs, coding patterns, and methodologies. It focuses strongly on business features in an application, with the author covering how to use OSGi, how to add authentication/security, and how to monetize from a modular application. You'll learn how to get started using the NetBeans platform with or without using the NetBeans IDE, how to set up a modular application, and more.

JAVA PROGRAMMING: A COMPREHENSIVE INTRODUCTION
By Herbert Schildt and Dale Skrien
McGraw-Hill Higher Education (December 2012)
Java Programming: A Comprehensive Introduction is designed for an introductory programming course using Java. This text takes a logical approach to the presentation of core topics, moving step-by-step from the basics to more-advanced material, with objects being introduced at the appropriate time. The book is divided into three parts. Part One covers the elements of the Java language and the fundamentals of programming; Part Two introduces graphical user interface programming using Swing; and Part Three explores key aspects of Java's API library.

PROGRAMMING CLOJURE, SECOND EDITION
By Stuart Halloway and Aaron Bedra
Pragmatic Bookshelf (April 2012)
Programming Clojure, Second Edition is a significant update to the classic book on the Clojure language. You'll get thorough coverage of all the new features of Clojure 1.3, and enjoy reorganized and rewritten chapters that reflect the significance of new Clojure concepts. Many code examples have been rewritten or replaced, and every page has been re-evaluated in light of Clojure 1.3. As the authors show you how to build an application from scratch, they provide a rich view into a complete Clojure workflow. You'll also get an education in thinking in Clojure.

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

DEVICE CONVERGENCE

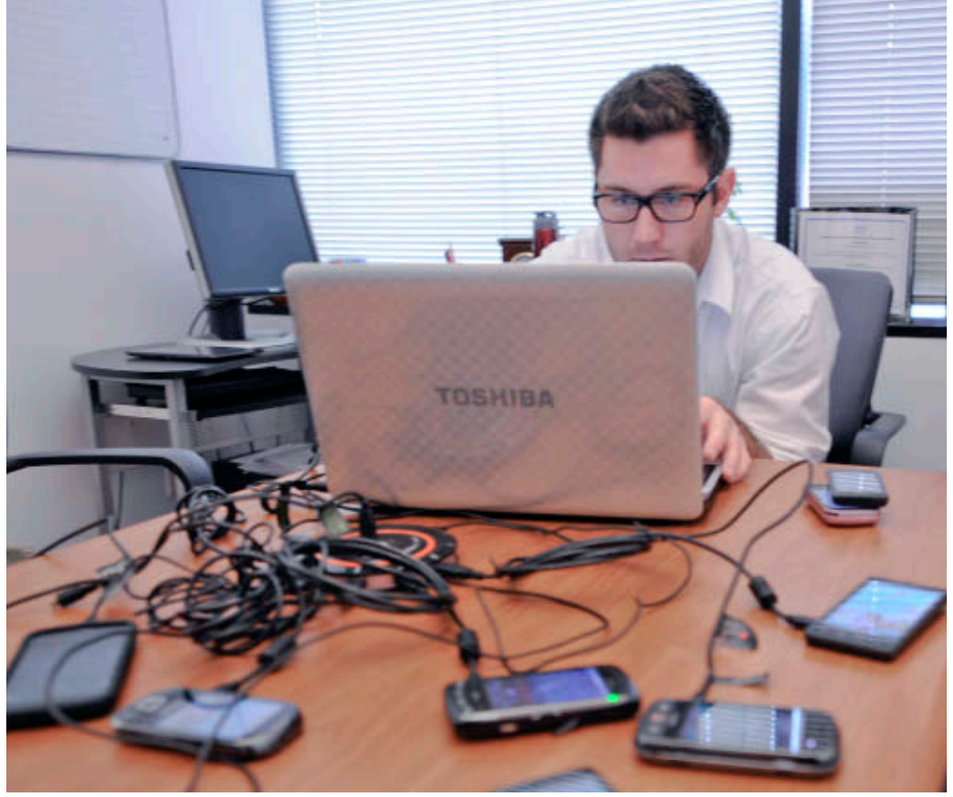
Jargon Technologies advances home entertainment frontiers with Java.

BY DAVID BAUM

Children's movies often contain powerful themes for adults—and sometimes a striking parallel to real life. So it was for Bhanu Srikanth, CEO at Jargon Technologies, a small company specializing in Java solutions for the media industry. Like the characters in the popular movies that her company works on—from *Rio* and *Ice Age* to *Alvin and the Chipmunks*—Srikanth's professional career has followed a series of twists and turns, a real-life "hero's journey" that has included unexpected sojourns in distant lands, overcoming a mounting series of obstacles, and, finally, discovering new professional horizons in a warm, hospitable climate.

Bhanu Srikanth, CEO, Jargon Technologies

PHOTOGRAPHY BY PHIL SALTONSTALL
ART BY I-HUA CHEN



Troy Mockenhaupt, senior engineer at Jargon, tests out various devices.

SNAPSHOT

JARGON TECHNOLOGIES
jargon-tech.com

Headquarters:
Burbank, California

Industry:
Media and entertainment

Employees:
15

Java version used:
Blu-ray Disc Java, based on Java ME CDC/PBP Packaged Media profile of Globally Executable MHP (GEM)

Srikanth grew up in India, where she also attended college and obtained a master's degree; she then worked in Japan helping an elite group of software engineers automate a steel mill. She transitioned into media technology in 2002 when she joined Panasonic Hollywood Laboratories (PHL) in Southern California, where she was part of the core R&D team that developed the Blu-ray Disc high-definition optical media format. It was an exciting time for the home entertainment industry. Along with Sony and Philips, Panasonic was working diligently to gain market acceptance against the rival HD DVD format backed by Toshiba. Srikanth represented Panasonic at several meetings of the Blu-ray Disc Association and also worked with some of the Hollywood studios to create Blu-ray content.

As consumer interest in high-definition video content increased, Srikanth and her colleagues saw an opportunity to strike out on their own.

"We started seeing a need for Blu-ray expertise that was lacking in the industry, and there were only a few companies that were able to do this," she recalls. "We founded Jargon Technologies to realize the vision of the creative community in Hollywood, as well as in other industries such as education."

BIRDS OF A FEATHER

Jargon's other founding members included Jeff Schulz, chief platform architect, and Nathan Epstein, COO. Srikanth donned the mantle of CEO, taking responsibility for account management, business development, and overall corporate strategy. The new company soon landed high-profile projects with Panasonic, Walt Disney Pictures, Warner Brothers, and Twentieth Century Fox Home Entertainment. Its forte was developing interactive applications that reside on the bonus discs distributed with popular Blu-ray titles.

"Slowly, we started doing more and more Blu-ray work, but we also branched out into connected TV, mobile, and other areas," says Srikanth. "Our goal is to reach a wide audience through innovative use of Java technology—specifically, to help special-needs children realize their full potential."

The market reached a tipping point in 2008 when several studios and distributors shifted their allegiance to the Blu-ray format. On February 19, 2008, Toshiba officially announced that it would stop the development of HD DVD players. Blu-ray had won the format wars, and, almost overnight, Jargon found itself at the center of a revolution in interactive entertainment.

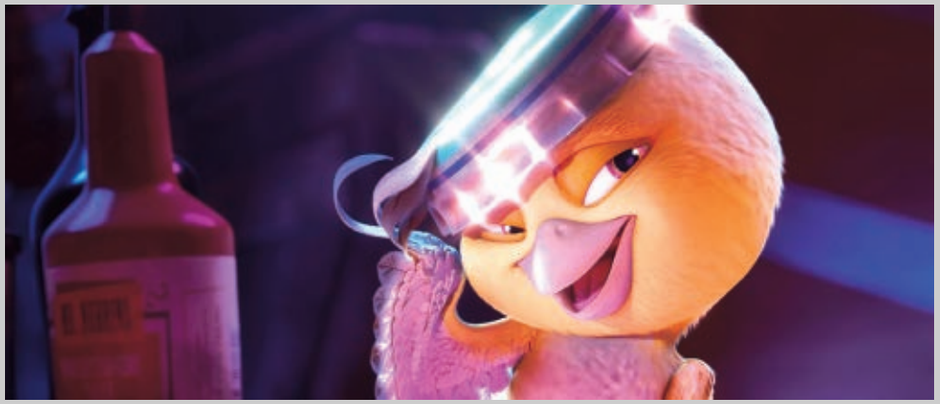
"From the outset, we saw this opportunity as much more than just content for Blu-ray players," Srikanth says. "We were thinking about communications, about home networks, and about connected devices. We said, 'Let's see how much Blu-ray can do. . . .'"

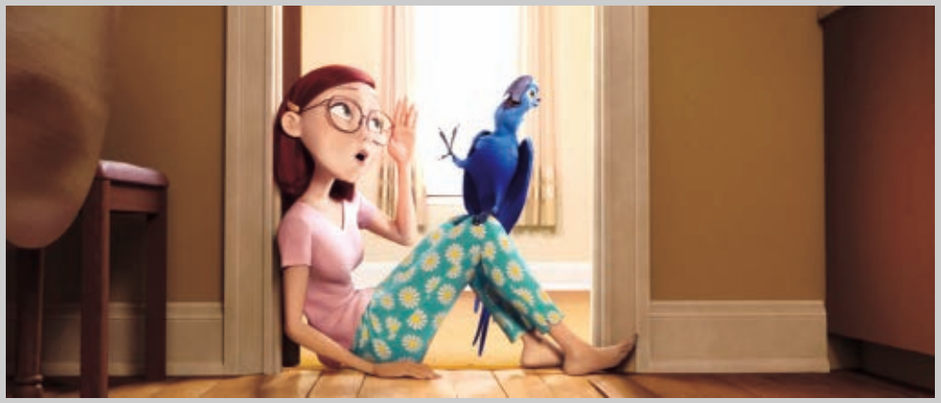
TAKING FLIGHT

Be careful what you wish for. It was around this time that Twentieth Century Fox Home Entertainment asked Jargon to implement a new type of game for the Blu-ray release of the movie *Rio*, a very successful animated film about a domesticated macaw from small-town Minnesota who takes off on an adventure to Rio de Janeiro, Brazil. The theatrical release had been a worldwide success, with box office earnings nearing US$500 million dollars. Now it was time to bring the family adventure to the cable and home entertainment market.

Jargon's technology team brainstormed with Fox Home Entertainment's content directors about ways to push the boundaries of Blu-ray Disc Java (BD-J) to enhance the consumer

THE POWER OF BD-J
Blu-ray Disc Java powers bonus view content including network access, picture-in-picture, and local storage.





POSTCARDS FROM RIO
***Rio* on Blu-ray viewers can create postcards using scenes, images, and words from the movie and then display, print, or send them to another device.**

RIO IMAGES COURTESY OF TWENTIETH CENTURY FOX

experience. They came up with the idea of an interactive game called "Postcards from Rio" that lets viewers create postcards using scenes, images, and words from the movie.

"We had this idea of letting viewers create postcards by grabbing frames from the movie, adorning the scene with stickers, and then adding bits of dialogue," explains Srikanth. "The program would format it like a real postcard and then let you display it, print it, or send it to another device."

Fox Home Entertainment provided the cinematic content based on memorable scenes in the movie, such as a beach theme and a carnival theme, along with sticker-like images of the characters, backgrounds, and props. Viewers could use their remote controls to pick a scene they liked and then adorn it with beach balls, umbrellas, headdresses, and other bits of movie memorabilia. The game was an immediate success.

"The amount of customization allowed here impresses, as does the fluidity of the exercise," noted one prominent DVD critic on his Website. "This may exceed every DVD set-top game I've seen in terms of enjoyability."

DEEP DIVE

Rio is bright, colorful, and full of surprises—much like the highly immersive applications that Jargon creates for Hollywood studios. To bring the images, icons, and bits of dialogue to life in an interactive game, Jargon used two standard Java technologies: BD-J and BD-Live.

BD-J is a specification for creating advanced content on Blu-ray Discs—it's what makes Blu-ray Disc titles so much more sophisticated than similar content on a standard DVD player, including network connectivity, picture-in-picture (PIP), and access to local memory storage. In this case, it lets *Rio* viewers capture, resize, rotate, and position their

TOP TITLE
Thanks in part to its innovative interactive content, Rio reigned as the #1 Blu-ray on retail charts for four weeks in a row.

Jargon's Srikanth, Mockenhaupt (center), and Nathan Epstein, COO, test out a new interactive program.

chosen images, and then add text and stickers from an onboard content library.

BD-Live is a Blu-ray platform for network connectivity. It uses Java ME Connected Device Configuration/Personal Basis Profile (CDC/PBP) network APIs to enable BD-J applications to utilize internet connectivity for interactive experiences such as multiplayer games as well as to provide additional content for a particular disc. In this case, it lets *Rio* viewers send the postcards directly to their home printer and access them on their PC, Mac, tablet, and mobile phone.

Like Blu and Jewel, the title characters in the movie, Jargon was moving through uncharted territory, which presented a series of technical hurdles. The first one involved figuring out how to encode the completed postcards to a file for local storage. The second entailed sending the file to a printer on the viewer's home network. And the third required figuring out how to share the postcard with other wireless devices, such as tablets and mobile phones.

To surmount the first two hurdles, Jargon implemented a DLNA printer discovery and printing layer along with methods to access the postcard using BD-J. Led by Jargon founding members Schulz and Epstein, their efforts resulted in the first Blu-ray application to utilize BD-J to connect and print directly to a consumer's home printer.

"We implemented a variety of technologies natively in Java as part of the BD-J application, including DLNA/UPnP, NetBIOS, and HTTP," Srikanth explains. "Our developers also figured out how to make a non-UPnP device into a UPnP device by writing the code on the Blu-ray Disc itself, since those technologies aren't necessarily resident on every Blu-ray player."

They surmounted the final obstacle by developing a new product called JargonTalk, a cross-platform technology that enables discovery and interaction of devices on a local network.

"We had to do some very tricky coding to be able to get the real IP address of each Blu-ray player and then make it into an HTTP Web server so it could broadcast that address to other devices," Srikanth continues. The team solved this problem by implementing NetBIOS in the BD-J application. Now, viewers simply type HTTP://rio from any Windows system to retrieve the postcards, or they can view them on any non-Windows device by referencing a URL displayed on the TV screen.

GETTING ENGAGED

The completed postcards can be accessed on any device, such as a smartphone, tablet, or PC, on the home network. In Hollywood parlance, this capability is known as *second screen*, referring to an independent device that displays interactive content related to the program being watched on the main screen and permits the user to become an active agent in the program. JargonTalk is gaining traction as an ideal way to facilitate these interactions.

"We call our technology 'second screen and beyond' because we are extending the experience not just to a second device, but also to multiple devices throughout the home," Srikanth says. "Thanks to Java, we have an unlimited set of deployment platforms, including devices like printers that don't even have an operating system."

The potential extends beyond the entertainment industry to include applications in medicine, event planning, telecommunications—just about any domain that requires multifaceted mobile communication.

"As a platform, Java is very flexible and adaptable," sums up Srikanth. "We were able to marry very divergent technologies like NetBIOS and UPnP and the HTTP protocols to make a Blu-ray player do something that it wasn't designed to do, resulting in a unique interactive experience. Java is the ideal platform for our needs. It's powerful. It resides in every conceivable electronic device. Java is everywhere." `</article>`

David Baum is a freelance business, technology, and lifestyle writer in Santa Barbara, California. .

For two developers, **Java ME** is the key to winning in India's mobile games market.

BY PHILIP J. GILL

Game On!

The "smallest" Java platform, Java ME, is playing a major role in helping mobile application developers profit in one of the world's largest and fastest-growing economies: India. With 1.2 billion people, India is the world's second-most-populous country, behind China. Not surprisingly, this South Asia giant is also the world's second-largest mobile

PHOTOGRAPHY BY NAMAS BHOJANI

Top: Twist Mobile CEO Virat Khutal; Bottom: Nextwave Multimedia CEO P.R. Rajendran (left) and COO P.R. Jayshree

Members of the Nextwave Multimedia team show off their latest mobile apps.

SNAPSHOT

NEXTWAVE MULTIMEDIA
nextwavemultimedia.com

Location:
Chennai, Tamil Nadu, India

Industry:
Software

Employees:
70

Java version used:
Java ME
Standard Edition 6

phone market—again behind China—with more than 884 million subscribers, or 73 percent of its population. Java ME is the natural choice for mobile app developers in India. This Java standard, designed specifically for mobile phones, PDAs, and other embedded systems, has become a market standard in India and around the world. Indeed, worldwide there are more than 2.1 billion Java ME–enabled mobile phones and PDAs. To understand the role of Java ME in the Indian mobile apps market, *Java Magazine* spoke with two award-winning developers about the big opportunities Java's smallest platform has opened up for them.

CATCHING THE WAVE

Every savvy developer hopes to get in on the ground floor of the next "hot" application category. So it was for Nextwave Multimedia, a 10-year-old IT services firm in Chennai, the capital of the Indian state of Tamil Nadu. Nextwave originally concentrated on developing applications for local advertising and media firms, and over time expanded into custom and mobile apps for both national and international clients, explains Nextwave CEO P.R. Rajendran—who founded the company with his sister, P.R. Jayshree, who now serves as COO.

GROWTH SPURT
With more than 884 million subscribers, India is the world's second-largest mobile phone market.

Two years ago, the successful outsourcing firm switched gears from developing custom apps for outsourcing clients in the U.S. and Europe to building and marketing its own mobile apps for India's rapidly expanding mobile phone market. "We felt that with 8 years of experience, we had enough maturity and understanding of the marketplace to build our own products," explains Rajendran. "Many years of experience devel-

oping for demanding clients gave us the confidence to get into developing our own intellectual property."

Mobile apps were a good fit for the company's first products. For one thing, the company had considerable experience in developing mobile apps for others. "We have developed custom mobile apps for 15 of the Fortune 500," Rajendran notes. For another, the mobile phone market in India was exploding, particularly for lower-priced "feature phones" that in India can cost anywhere from US$20 to US$100.

Another good fit was Java ME. "Java ME is the default choice," explains Rajendran. "India has a very large number of affordable handsets that support Java technology. Java ME has the most-efficient toolsets for developing content for these affordable devices."

Today the company offers a catalog of 80 apps, mostly games and entertainment, available for free download from its Website. Among its most popular apps is Comics Creator.

"This tool allows you to very quickly create comics on your mobile phone while on the go," says Rajendran. "Comics Creator has a library of characters, and the characters have many different facial expressions. There's also a library of backgrounds and verbal expressions. The idea is that the user can put these all together to tell a story."

"Using characters, backgrounds, and speech vocals, Comics Creator can be used to make a comment on a social issue or a joke," he continues. "The comics format is very compelling and interesting."

Comics Creator has also proven quite popular. Recently, Nextwave won an award from Nokia, the largest provider of mobile phone handsets in India, for passing more than 1 million downloads from Nokia's online app store, Nokia Store. As a result, Nextwave was inducted into the Nokia Millionaire League (see sidebar, "The Millionaire League").

"Nokia's Java-enabled phones are our big focus, and in India Nokia is the biggest player in the market. Its feature phones are

SNAPSHOT

TWIST MOBILE
twistmobile.in

Location:
Indore, Madhya Pradesh, India

Industry:
Software

Employees:
42

Revenue:
US$365,000

Java version used:
Java ME
Standard Edition 6

Top: Twist Mobile developers test out the company's latest app. Bottom: Members of the Twist Mobile team brainstorm about a new game concept.

The Millionaire League

"The success of Indian app developers crossing several million downloads on Nokia Store is a reflection of India's inevitable progress to becoming an app superpower," says Gerard Rego, director, Ecosystem and Developer Experience, at Nokia India. "As a leader in the industry," Rego continues, "Nokia is committed to nurturing a culture of innovation in the mobile app industry and supporting developers with tools and insights that can help them develop successful and rewarding applications for their consumers."

Nextwave Multimedia and Twist Mobile are both members of the Nokia Millionaire League, a select club of 350 Nokia mobile application developers from around the world. This exclusive club is reserved for mobile app developers who have had more than 1 million downloads from Nokia's online app store, Nokia Store.

Nokia executives and technologists see the mobile application industry as one of the fastest-growing sectors in the world and believe that some of the most-talented developers come from India. India is among the top three markets for application downloads from Nokia Store, and with 80 percent of its traffic converting to downloads, Nokia Store is one of the most promising stores for developers and consumers. There are more than 200,000 registered Nokia developers in India, the single largest concentration from any country.

In fact, of the new members inducted into the global Millionaire League last year, many—including Nextwave and Twist Mobile—are from India. Nextwave has had more than 25 million downloads of its mobile apps downloaded from Nokia Store, while Twist Mobile says its downloads are approaching 50 million.

dominating the market right now," says Rajendran. "Of the 30 million apps that have been downloaded, 25 million were downloaded by Nokia phones running Java ME."

A DIFFERENT TWIST

Virat Khutal's path to becoming a developer of Java-based mobile gaming applications wasn't a straight one. Khutal, a founder and CEO of Twist Mobile, located in Indore, the largest city in the central Indian state of Madhya Pradesh, completed an architecture degree before discovering Java. His first job as a developer was at a financial services company.

"I started working with Java about 10 years ago," says Khutal. "I was working at a financial services firm for about six or seven months, and they were looking for someone to write mobile applications, so I took a Java book home one evening at about 7 p.m. In the morning, I was actually able to start writing Java code. It was that easy for me, as well as many other people in my company. That's one of the attractions to Java."

After learning all he could about Java, Khutal eventually left the financial services company and started Twist Mobile. Given the nature of the Indian mobile applications market, Khutal says he decided to concentrate on gaming applications designed for lower-end feature phones. "At the time, the dominant handset maker was and continues to this day to be Nokia, whose most popular software platform is Series 40, a Java-based software platform that runs on some of the company's most successful handsets."

IT'S A NATURAL

Java ME is the natural choice for mobile app developers in India, such as Nextwave Multimedia and Twist Mobile.

Today, Twist Mobile offers 30 different mobile apps, mainly games. Among the company's most popular mobile apps is Age Effect. "The app allows users to download a photo of an individual and alter his or her face to see how he or she might look 30 years from now," says Khutal.

Other popular apps provide similar capabilities that allow users to alter photos—including Sketch Effects, which allows users to convert photos on their mobile phones to a sketch and then post them on Facebook; Photo Effect, which allows users to add special effects to photos on their phones; and Color Effect, which allows users to mix black-and-white and color images.

Khutal says that in a typical month, the company sees 5 to 6 million downloads, reaching an aggregate of almost 50 million downloads since the company's founding in 2009. Like most Indian mobile app developers, Twist Mobile doesn't charge the user when buying these apps; instead "we monetize our apps through in-app advertising," explains Khutal. "Right now in-app advertising generates about [US]$1,000 per day in income."

Khutal says that Java was the logical choice as a programming environment for mobile apps in general and the Indian mobile market in particular, given that Nokia is the dominant handset maker here and the Nokia Series 40 is based on the Java ME standard. "In India we have a huge population that can't afford very expensive handsets," explains Khutal. "Nokia is bringing the price down with advanced, high-quality features that make these feature phones more and more like smartphones."

He continues, "Java will always play a role in emerging markets such as India, because it is an open source technology that is always evolving and staying relevant to us." `</article>`

Philip J. Gill is a San Diego, California–based freelance writer and editor.

MICHAEL KÖLLING

BIO

The End of the Game

Let's finish our game and share it with the world.

The game we continued building in the previous issue of *Java Magazine* is getting nearer to completion, but some parts are still missing. Today, I'll show you how to finish the game and—most importantly—how to share it with your friends. If you haven't been playing along, you can download Greenfoot and our project and join in the fun.

Ending the Game

Currently, there is no way to win this game. Eat all the pizza slices, and nothing happens. No fanfare, no big "YOU WIN!" sign. Equally, when the snakes catch you, the game just quietly runs on.

We want to fix both those situations. Let's say that when you manage to eat 12 slices of pizza, you win. The game stops, and we hear a victorious fanfare. However, when the snakes catch you, you lose. We get a rather sad losing sound, and the game stops.

The second scenario (losing) is easier to do—we have almost everything for that already. So let's do that first, and then look at the functionality for winning.

Losing (Being Eaten by Snakes)

We already have a sound effect for when the snake eats the turtle (in the Snake's eat method), which uses Greenfoot's playSound method. The first thing we do is just change the sound to a new sound file (which is included with the project download) named trombone.wav.

Then we add a new line: a call to Greenfoot's stop method to stop the game at this point. Together, the code looks like this:

```
Greenfoot.playSound
("trombone.wav");
Greenfoot.stop();
```

Note that both methods are static methods of the Greenfoot class, so we have to write the class name Greenfoot and a dot (full stop) before the method name in the method call.

To discover what other methods the Greenfoot class offers, choose **Greenfoot Class Documentation** from the **Help** menu.

Alternatively, you can use *code completion*. For example, after typing Greenfoot. (that is, after the dot), press **Ctrl-Space** on your keyboard, and you will see a list of all methods that you can call at this point, as shown in **Figure 1**. Starting to type a method name will narrow the choices. Then choose any of the methods from the list.

Implementing this end to the game was easy enough. Doing a similar thing for winning is a bit more work.

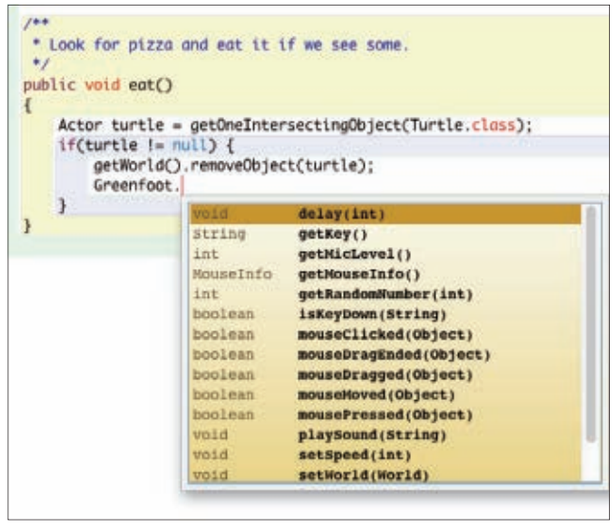


Figure 1

Winning (Finishing Your Pizza)

We want our winning functionality (fanfare and stopping the game) to occur when we have eaten our whole pizza (12 slices).

In choosing this criterion, we have two problems. First, we currently have only 7 slices of pizza in our game. We could, of course, simply put 12 slices in from the start, but that would make the game too easy. Instead, I want to make a new slice appear each time one is eaten.

And secondly, we now need to count how much we have eaten, so we know when to stop. We don't know how to do that yet.

Let's do these two things.

The Magically Appearing Pizza

The pizza slices get eaten in the Turtle class—to be specific, in the turtle's eat method. So this is where we start to work. To keep the code clean, we will place the instructions to create a new pizza slice in a separate method and call that method from here. So we place a call to placeNewPizza() into our eat method, just below the playing of the sound:

PHOTOGRAPH BY JOHN BLYTHE

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

The placeNewPizza() method will place a new slice into the world every time a slice gets eaten. This method does not exist—we will have to write it now.

The full code for this method is shown in **Listing 1**. We see the eat() method that calls the placeNewPizza() method and the definition for this new method. Let's analyze this line by line.

To place a new pizza slice at a random location into the world, we have to achieve three things:

- Create a new pizza object.
- Generate random coordinates.
- Place the pizza at those coordinates into the world.

The first line in our new method reads

```
World myWorld = getWorld();
```

We have seen before that the getWorld() call gives us a reference to our current world object. This time, we declare a local variable named myWorld (it is of type World) and store the world object in it. This, in itself, does not achieve very much yet, apart from the fact that we can now use the myWorld variable to talk to our world.

In the next two lines, we generate the random coordinates and store them in two local integer variables:

```
int x = Greenfoot.getRandomNumber
(myWorld.getWidth());
int y = Greenfoot.getRandomNumber
(myWorld.getHeight());
```

We have seen method calls to the getRandomNumber, getWidth, and getHeight methods before. Here, we generate a random number between 0 and the world's width and store it in the x variable. Then we do the same for y using the height of the world as the limit. This gives us random *x* and *y* coordinates that match the exact size of the world.

The last line then does the magic:

```
myWorld.addObject(new Pizza(), x,
y);
```

Here, we place the new pizza object into the world. To do this, we are using the world's addObject method. This method expects three parameters: the object to be placed, the *x* coordinate, and the *y* coordinate.

We can write new Pizza() to create a new object of type Pizza, and we can use this in place of the object to be placed into the world. Then we can use our x and y variables (which we have just filled with our random values) in place of the coordinates.

Read the complete **Listing 1** again and make sure that you understand it. Then try it out for yourself.

Counting Pizza

The functionality so far gives us an endless supply of pizza slices. The next task to be done is to count how much we have eaten. We do this by declaring a *field* (also known as an *instance variable*) named slices to use for the counting. (We're still working in the Turtle class.)

LISTING 1

```
/**
 * 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");
    placeNewPizza();
  }
}

/**
 * Place a new pizza slice at a random location into the world.
 */
public void placeNewPizza()
{
  World myWorld = getWorld();
  int x = Greenfoot.getRandomNumber(myWorld.getWidth());
  int y = Greenfoot.getRandomNumber(myWorld.getHeight());
  myWorld.addObject(new Pizza(), x, y);
}
```

Download all listings in this issue as text

This is what it looks like:

```
public class Turtle extends Actor
{
  private int slices = 0;

  ...
}
```

The field declaration consists of the keyword private, followed by a type and name for our variable, and an optional initialization value. The type and name (int slices) is the same as we have seen for a local variable. The initialization (= 0) assigns zero to the variable to start.

The placement of this line is important. Field declarations must be outside of method definitions, so we write this line immediately after the class header (before our act method).

The slices field can store integers (that is, whole numbers). It currently holds zero. We can now use this field to count our pizza slices. Every time we eat some pizza, we increment this field by 1. When we have reached 12, we play the fanfare (the sound file fanfare.wav, which is

included in the project download) and stop the game.

Listing 2 shows the complete implementation. Let's briefly examine the interesting aspects.

We have added the following line:

```
slices++;
```

The ++ operator, written after an int variable, increments (that is, counts up) this variable by 1. So we can see that we are counting up every time after removing a pizza slice from the world.

Next, we have the following lines (with some instructions in the place of the three dots):

```
if(slices == 12) {
  ...
}
else {
  ...
}
```

This code checks whether the slices variable is now equal to 12 and, if it is, executes the instructions between the first pair of curly brackets. Otherwise (if the variable is not 12), the instructions after the else keyword are executed. Note that we need to use a double equal symbol (==) to check for equality.

Read **Listing 2** carefully. You will see that this code is used to play our slurping sound and place a new pizza slice as long as we have not eaten 12 slices yet. Then it plays the fanfare and ends the game when we have eaten 12 slices. Try the code out in your own version of the game.

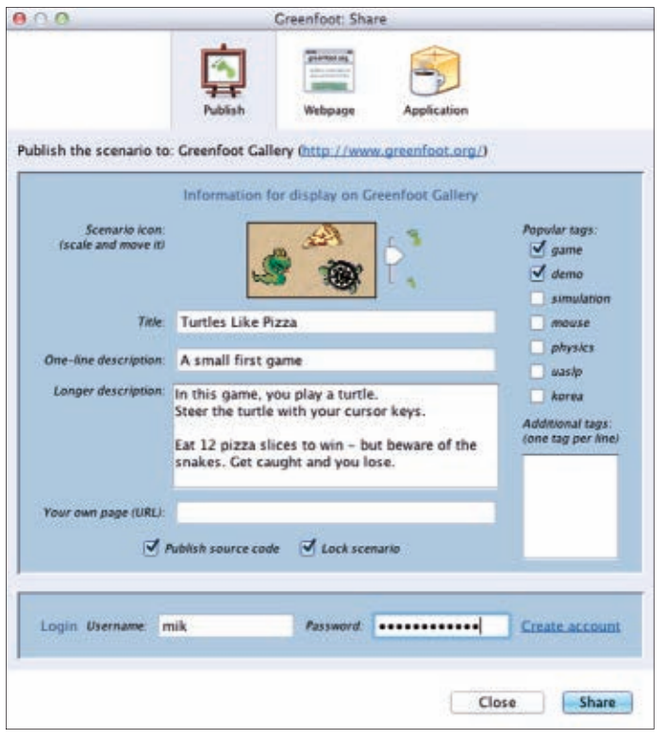


Figure 2

Sharing Your Game

Now that our game is "complete" in some sense—of course, no game is ever truly finished—it would be great if we could get all our friends to try it out. Luckily, we can.

Greenfoot has a "share" function that allows you to upload your program to the Greenfoot Website, where other users can play it and give you some feedback. You can see all the scenarios other Greenfoot users have created by going to the Greenfoot site and looking around.

To upload your own game, click the **Share** button in the top right of Greenfoot's main window. Clicking it brings up a dialog box that allows you to specify some details for your upload (see **Figure 2**).

For the upload to work, you must have an account on the Greenfoot site. (If you don't, click the **Create account** link.) You can browse and play other people's games without signing up, but to upload your own game, you need an account.

Once you have an account, this dialog box lets you add a title, a description, an icon, and some tags, and then you can upload your game. When you upload a game, you can choose whether you want to upload the source code, too. If you do, others can look at your source code and possibly learn from it or modify it themselves.

Your project will have a unique URL on the Greenfoot Website. Send this to your friends to show them the cool things you have done!

Conclusion

With this article, we have finished building a simple computer game using Greenfoot and Java. If you followed along throughout this series, you will have seen that building a simple game in Java is not very hard. And you can learn important programming concepts along the way. Now take a look around the Greenfoot site for some more ideas, and continue to build your own games. And above all, have fun! `</article>`

LISTING 2

```
/**
 * Look for pizza and eat it if we see some.
 */
public void eat()
{
  Actor pizza = getOneIntersectingObject(Pizza.class);
  if(pizza != null) {
    getWorld().removeObject(pizza);
    slices++;
    if(slices == 12) {
      Greenfoot.playSound("fanfare.wav");
      Greenfoot.stop();
    }
    else {
      Greenfoot.playSound("slurp.wav");
      placeNewPizza();
    }
  }
}
```

Download all listings in this issue as text

RÉGINA TEN BRUGGENCATE AND **LINDA** VAN DER PAL

BIO

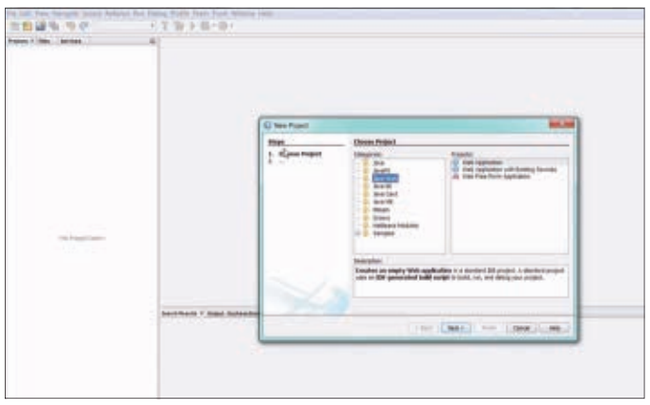
Build a Web App in Wicket 1.4 Using NetBeans

Combine Wicket and Java EE 6 to build a Web app easily by separating markup from business logic.

In this tutorial, we will show you how to build a simple Web application in Apache Wicket 1.4 using the NetBeans IDE. Wicket is a component-based Web application framework that lets you build enterprise-grade Web applications using Plain Old Java Objects (POJOs), HTML, Ajax, Spring, Hibernate, and Maven.

We chose Wicket as our subject because we love its separation of markup and logic and the way you can build components that you can reuse. We also wanted to show how it can be combined with Java EE 6, because Java EE 6 is gaining more and more momentum and is quite easy to learn. And we used NetBeans because it offers good integration between the two.

We would have liked to show you how to build an entire Web shop with Wicket. But you'll just have to read *Wicket in Action* for that. All we have space for in this article is to show you how to display a list of products with a link to a details page. We'll build the Web app gradually and show you some alternatives along the way. **Note:** The source code and other necessary files for the project shown in this tutorial can be downloaded here (note: 14.6 MB).



Watch the Set Up Project movie to see the steps to create the project.

Setting Up Your Project

The first part of this tutorial is based on Jeff Schwartz' blog about Java EE 6 and Wicket. If our summary is too concise for your purposes, read the series of articles mentioned in Schwartz' blog.

Downloading the required software and files. In this section, you are going to set up your development environment.

1. Download and install the following software: MySQL, NetBeans, and the Wicket plug-in for NetBeans.
2. You will use create_script_mysql.sql to create the backing database. Execute this script in the command line tool by running the following line: \.Create_script_mysql.sql.

Creating the project. In this section, you will create the project and create the default Wicket application that will be modified in a later section.

1. In the NetBeans IDE, from the **File** menu, select **New Project**.
2. Select **Java Web** in the Categories panel, select **Web Application** in the Projects panel, and click **Next**.
3. Enter DuchessStore in the **Project Name** field, and click **Next**.
4. Make sure **GlassFish Server 3** is selected as the server and **Java EE 6 Web** is selected as the Java EE version, and click **Next**.
5. In the Frameworks panel, select **Wicket**. (If Wicket isn't available as a framework, the Wicket plug-in wasn't installed correctly.)
6. Enter org.duchess.example.wicket for **Main Package**, and click **Finish**.

Note: To see these steps in action, view the Set Up Project movie.

Adding the Java EE Inject library. Wicket doesn't have Java EE support included, but there is a library that adds this functionality. More information on this library can be found on GitHub.

Because this project was built to work with Maven, but we're not using Maven for this project, Schwartz was kind enough to find out which JAR files were needed. He provided them to us in a zip file, which we are now passing on to you.

1. Open the libraries.zip file and install the three JAR files it contains into your project. For more information, view the Add Java EE Inject Library movie.
2. In order to make these JAR files do their trick, add the method shown in **Listing 1** to the Application.java file.

Creating the entities. In this section, you will create the entities that will represent the data from the database. In the older versions of Java EE, these were called EJBs.

1. In the NetBeans IDE, from the **File** menu, select **New File**.
2. Select **Persistence** in the Categories list.
3. Select **Entity Classes from Database** in the File Types list, and click **Next**.
4. Select **New Data Source** from the Data Source list.
5. Enter duchess_store in the **JNDI Name** field and select **New Database Connection** from the Database Connection list.
6. In the Driver list, select **MySQL (Connector/J Driver)**, and click **Next**.
7. Enter duchess_store in the **Database** field, enter the username and password you created when you installed MySQL, and click **Finish**.
8. Click **Add All** to add all tables as entities, and click **Next**.
9. Enter org.duchess.example.entities in the **Package** field, and click **Next**.
10. Select **java.util.List** from the Collection Type list, and click **Finish**.

Note: To see these steps in action, view the Create Entities movie.

Hint: If you ever need to delete the database connection, you have to do it in both the glassfish-resources.xml file (delete jdbc-connection-pool and jdbc-resource) and under the databases in the **Services** tab. If this still doesn't remove the connection, start the server from the **Services** tab, look under **Servers -> GlassFish Server -> Resources -> JDBC -> JDBC Resources**, and unregister the connection there.

GOOD STUFF

NetBeans offers good integration between Java EE 6 and Wicket.

Creating the session beans. In this section, you will create the session beans that will contain the business logic of the application with regard to the handling of data to and from the database.

1. In the NetBeans IDE, from the **File** menu, select **New File**.
2. Select **Persistence** in the Categories list.
3. Select **Session Beans For Entity Classes** in the File Types list, and click **Next**.
4. Click **Add All**, and click **Next**.
5. Enter org.duchess.example.beans in the **Package** field, and click **Finish**.

Note: To see these steps in action, view the Create Session Beans movie.

LISTING 1 LISTING 2 / LISTING 3

```
@Override
protected void init() {
 super.init();
 addComponentInstantiationListener(new JavaEEComponentInjector(this));
}
```

Download all listings in this issue as text

Getting Started with Wicket

Congratulations; you now have a running Wicket project. Of course it doesn't really do anything yet, but you're going to change that in this section. First, let's change the application so it shows a list of products that are in the database.

Making a list of products. To start, we'll add some images and a stylesheet, because we'll need those later.

1. Unzip the images.zip file into the wicket directory. If all is well, there should now be an images directory there.
2. Open the stylesheet.css file in the wicket directory as well.

Now we can get started on the code. Because we don't like the way the current page looks, we'll change the BasePage and the HeaderPanel and remove the FooterPanel entirely.

1. Look for the BasePage.html file in the org.duchess.example.wicket package and replace this with the code in **Listing 2**. This is the basis for all our HTML pages.
2. In the same package, you can find the BasePage.java file. Compare this with **Listing 3**. As you can see, we have removed the FooterPanel from the

BasePage and we are no longer passing a String to the HeaderPanel. Now replace the code with the new code, as shown in **Listing 3**. This will cause a compiler error, which we will fix in the next two steps.

3. In the org.duchess.example.wicket package, you can find the HeaderPanel.html file that needs to be replaced with the code shown in **Listing 4**. We have added a stylesheet, as well as an image with our logo, to the HeaderPanel. We have also changed the text that is shown. This will show on the top of all the pages that inherit from the BasePage.
4. And finally the code in the HeaderPanel.java file will be replaced with the code found in **Listing 5**. This will resolve the compiler error from Step 2.

Now we come to the real work. In order to show all the products from our database, we will use the simplest component Wicket offers to make a list of repeating items: the RepeatingView. This component renders all its children in the order in which they were returned from the database.

5. In the org.duchess.example.wicket package, you can find the HomePage.html file. This is the first page that is shown when the application is started. Replace the code with the code from **Listing 6**. We added a table to the page. We gave it some headers and a single row. This row has a wicket-id that we will use to couple this row to the RepeatingView. The row has three columns that contain the data we want to show.
6. In the same package, you can find the HomePage.java file. This code should be replaced with the code from **Listing 7**. In the HomePage class, we create a RepeatingView instance and give it the same ID as our row. We add this repeater to our page. Then we fetch all our products from the database and loop over them.

For each item, we create a WebMarkupContainer. As explained in *Wicket in Action*, WebMarkupContainer is a generic component that in itself doesn't do much. It can contain child components, so it is a handy tool to group components or use as an intermediate layer when you need to group more markup for a component. If we didn't use it here, we would get an exception when trying to add more than one product to our page, because the ID name is used more than once. If we run the application, the page now looks like **Figure 1**.

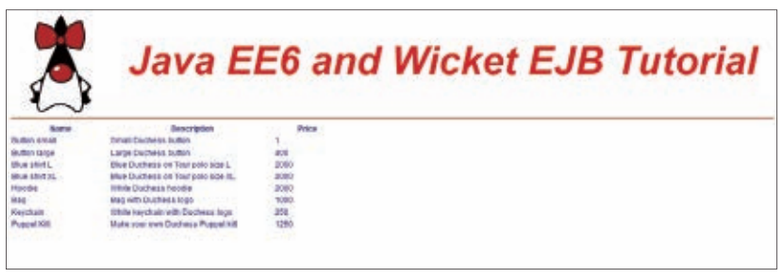


Figure 1

Adding paging. If our database ever grows to contain more products than would fit in a single screen, our solution wouldn't look very nice anymore. We would like to add paging to solve that. In order to do this, we will change the RepeatingView into a PageableListView.

1. We will first change the RepeatingView into a ListView. For this, only the HomePage.java file needs to be changed. The HTML file stays the same. As you can see in **Listing 8**, this code is a little bit simpler. We don't need to loop over the items ourselves anymore; the ListView does that for us. And we no longer need the WebMarkupContainer. All we have to do is override and implement the populateItem method. Now we replace

LISTING 4 LISTING 5 LISTING 6 LISTING 7 LISTING 8

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.
w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns:wicket="http://wicket.apache.org">
  <head>
    <link rel="stylesheet" type="text/css" href="style.css" />
  </head>
  <body>
    <wicket:panel>
      <wicket:link>
        <img src="images/duchess.jpg" style="float:left;"/>
      </wicket:link>
      <h1>
        <span wicket:id="headerpanneltext"/>
      </h1>
      <hr/>
    </wicket:panel>
  </body>
</html>
```

Download all listings in this issue as text

the code in the HomePage with the code shown in **Listing 8**.

The next step is to add actual paging. Once again, this is a relatively small step. But this time we need to change both the HomePage.java file (shown in **Listing 9**) and the HomePage.html file (shown in **Listing 10**).

2. As you can see in **Listing 9**, all we needed to do was change the type of the ListView and add a parameter to the constructor indicating how many elements we want to show in the list. We also needed to add a PagingNavigator, which will handle the navigation through our pages. So now we replace the code in the HomePage.java file with the code in **Listing 9**.
3. As you can see in **Listing 10**, the changes in the HTML file are even more minimal; we only needed to add the navigator markup. Now we add the highlighted code from **Listing 10** to the HomePage.html file.

Now our page looks like **Figure 2**.

When our list grows so large that we need many pages to show our products, this navigator doesn't scale very nicely. It shows at most ten pages and then doesn't inform us about how many more pages there are. It would be nice if we could show a different navigator when there are many pages and not show a navigator when there is only one page. These changes can be accomplished by creating a PagerFactory that will give us different navigators depending on the number of products. We'll also need to create our own navigators and some properties for those navigators.

4. First, create a new org.duchess.example.wicket.components.navigator package for the new navigator classes.
5. The first class you need to create is PagerFactory. The code for this class can be found in **Listing 11**. This code will cause several compiler errors, but those will be fixed later on.

The next steps will be for the pager that will be shown when there are more than five pages to navigate through.

6. Create a new Wicket Page named HighNumberNavigator.
7. Replace the code of the newly created HTML file with the code from **Listing 12**.
8. Replace the code of the newly created Java file with the code from **Listing 13**. You'll have to organize the imports and reformat the code to make it more readable, because we were constrained by formatting rules.

The next steps will be for the pager that will be shown when there are less than five pages to navigate through.

Figure 2

LISTING 9 LISTING 10 LISTING 11 LISTING 12 LISTING 13

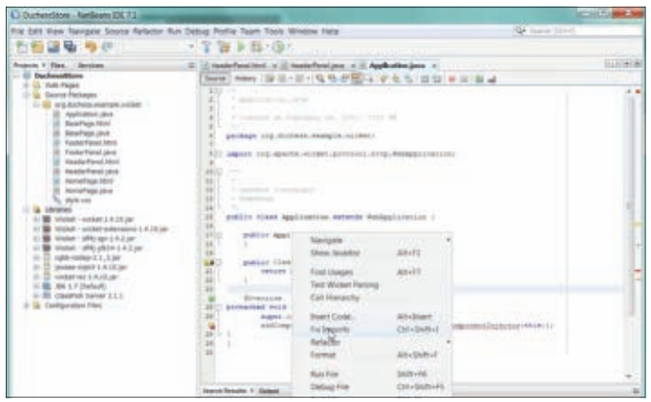
```
package org.duchess.example.wicket;

import java.util.List;
import javax.ejb.EJB;
import org.apache.wicket.markup.html.basic.Label;
import org.apache.wicket.markup.html.list.ListItem;
import org.apache.wicket.markup.html.list.PageableListView;
import org.apache.wicket.markup.html.navigation.paging.PagingNavigator;
import org.duchess.example.beans.ProductFacade;
import org.duchess.example.entities.Product;

public class HomePage extends BasePage {

  @EJB(name = "ProductFacade")
  private ProductFacade productFacade;

  public HomePage() {
    List<Product> products = productFacade.findAll();
    PageableListView<Product> productList =
        new PageableListView<Product>("productList", products, 5) {
      @Override
      protected void populateItem(ListItem<Product> item) {
        Product product = item.getModelObject();
        // name
        item.add(new Label("name", product.getName()));
        // description
        item.add(new Label("description", product.getDescription()));
        // price
        item.add(new Label("price", product.getPrice().toString()));
      }
    };
    add(productList);
    add(new PagingNavigator("navigator", productList));
  }
}
```

Download all listings in this issue as text

9. Create a new Wicket Page named LowNumberNavigator.
10. Replace the code of the newly created HTML file with the code from **Listing 14**.
11. Replace the code of the newly created Java file with the code from **Listing 15**. Once again, you'll need to organize the imports. After this step, the compiler errors of the PagerFactory should be solved.
12. For the text in the pagers you'll need to add a properties file called application.properties in the org.duchess.example.wicket package. Replace the content of the new file with the content of **Listing 16**.
13. In the HomePage.java file, you'll need to edit the code as shown in **Listing 17**.
14. In the HomePage.html file, you'll need to edit the code as shown in **Listing 18**.

BUILDING MADE EASY

Wicket is a **component-based Web application framework** that lets you build enterprise-grade Web applications easily.



Watch the Add Java EE Inject Library movie to learn about this Wicket library.

Creating a DataProvider. When the list of products gets too large, using a list as the model becomes burdensome. Even the PageableListView loads the whole list in memory before rendering the items on the page.

To overcome this problem, we need to use a DataView, which uses an implementation of IDataProvider to act as an interface between the database and the data view.

Our first step for implementing a DataView is to create a data provider.

1. Create a package called org.duchess.example.wicket.dataproviders.
2. Create an interface in this package called IEjbDataProvider.
3. Fill the interface with the code shown in **Listing 19**.
4. Create a class in this package called ProductDataProvider.
5. Fill the class with the code shown in **Listing 20**, and organize the imports.

After creating the data provider, we'll finish our list of products by putting the list in its own Panel. The advantage of doing this is that we'll be able to reuse or move the list easily later on, if we might want to. A Panel is a Wicket component that has its own markup. So we'll need to create both ProductsPanel.java and ProductsPanel.html.

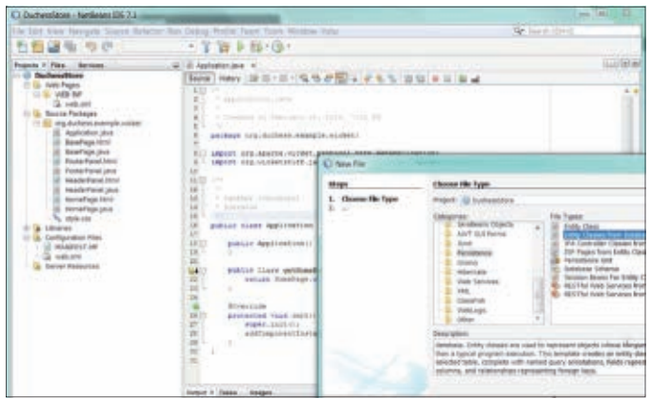
LISTING 14 LISTING 15 LISTING 16 LISTING 17 LISTING 18 LISTING 19

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
  xmlns:wicket="http://wicket.apache.org/">
<body>
<wicket:panel>

  <div class="box default">
    <div class="pages">
      <a wicket:id="first" style="visibility:hidden"></a>
      <a wicket:id="last" style="visibility:hidden"></a>
      <a wicket:id="prev" class="back">
        <wicket:message key="page.generic.navigator.back"/>
      </a>
      <a wicket:id="next" class="forward">
        <wicket:message key="page.generic.navigator.forward"/>
      </a>
      <span wicket:id="navigation">
        <a wicket:id="pageLink" href="#">
              <span wicket:id="pageNumber">5</span>
            </a>
      </span>
    </div>
  </div>
</wicket:panel>
</body>
</html>
```

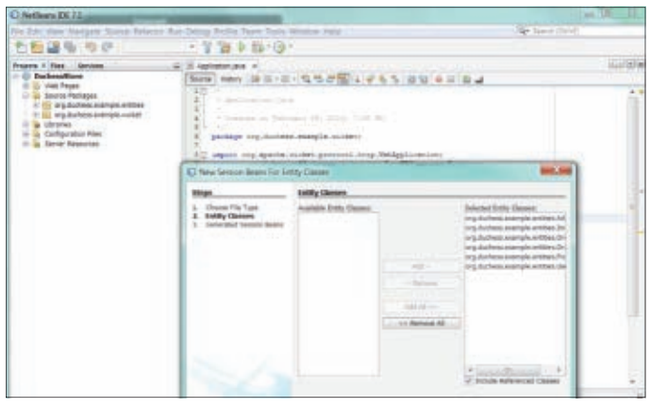
Download all listings in this issue as text

Fortunately, we can just have NetBeans do this for us.

1. Right-click the wicket package.
2. From the **New** menu, select **Java Package**.
3. Enter org.duchess.example.wicket.product in the **Package** field, and click **Finish**.
4. Right-click the new product package.
5. From the **New** menu, select **Other**.
6. Select **Web** in the Categories list.
7. Select **Wicket Panel** in the File Types list, and click **Next**.
8. Enter ProductsPanel in the **File Name** field, and click **Finish**.
9. Fill the ProductsPanel.html file with the code shown in **Listing 21**.
10. Fill the ProductsPanel.java file with the code shown in **Listing 22**.
11. And finally, add the new Panel to the HomePage by changing both the HomePage.java file (**Listing 23**) and the HomePage.html file (**Listing 24**).



Watch the Create Entities movie to see the steps to create entities.



Watch the Create Session Beans movie to see the steps to create session beans.

Creating a second page to show the details of a product. Next, we want to create a page where we can show the details of a product. First, we need to create a ProductDetailPage:

1. Right-click the product package.
2. From the **New** menu, select **Other**.
3. Select **Web** in the Categories list.
4. Select **Wicket Page** in the File Types list, and click **Next**.
5. Enter ProductDetailPage in the **File Name** field, and click **Finish**.
6. Fill the ProductDetailPage .html file with the code shown in **Listing 25**. And format the code to make it more readable.
7. Fill the ProductDetailPage .java file with the code shown in **Listing 26**. And organize the imports.
8. Create a class called ImagesView in the product package.
9. Fill the ImagesView.java file with the code shown in **Listing 27**.

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

```
package org.duchess.example.wicket.dataproviders;

// imports

abstract public class ProductDataProvider implements IEjbDataProvider<Product> {

  @Override
  public Iterator<? extends Product> iterator(int first, int count) {
    // to make sure we are not going to get a negative number of values
    // to retrieve, we need to reset the count
    if (count < first) {
      count += first;
    }
    int[] range = {first, count};
    List<Product> products = getFacade().findRange(range);
    return products.iterator();
  }

  @Override
  public IModel<Product> model(final Product object) {
    final Long id = object.getProductId();
    LoadableDetachableModel<Product> ldm = new
      LoadableDetachableModel<Product>(object) {
      @Override
      protected Product load() {
        return getFacade().find(id);
      }
    };
    return ldm;
  }

  @Override
  public int size() {
    return getFacade().count();
  }

  @Override
  public void detach() {
  }
}
```

Download all listings in this issue as text

10. Now, add a link for each product in the productList on the ProductsPanel. To do this, change both the Java and HTML files. **Listing 28** shows the ProductsPanel.html file, and **Listing 29** shows part of the ProductsPanel.java file. We only need to replace the highlighted bit of code.
11. Finally, unzip the images_products.zip in the directory org\duchess\example\wicket\product and run the insert_images.sql script to complete the application.

Conclusion

We'd have loved to show you how to do things like register a user, log on and log off, show user profiles, and so on. (Have a look at our final project to see all the code we wanted to showcase.) But we'd like to think that we've shown you a little of what you can do with Wicket and how easy it is to build a Web application with Wicket and NetBeans. `</article>`

LEARN MORE

- Apache Wicket

LISTING 25 LISTING 26 LISTING 27 LISTING 28 LISTING 29

```
<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE html>
<html xmlns:wicket="http://wicket.apache.org">
 <head><wicket:head><title>Duchess store</title>
  <wicket:link><link rel="stylesheet" type="text/css" href="style.css"/>
</wicket:link></wicket:head></head>
 <body><wicket:extend>
   <form method="post" action="#" id="product-form" wicket:id="productForm">
    <table>
     <tr>
      <td><label wicket:id="page.productId" for="productId"/></td>
      <td><input wicket:id="productId" type="text" maxlength="10"
         name="" id="productId" class="text size6"/></td>
     </tr><tr>
      <td><label wicket:id="page.name" for="name"/></td>
      <td><input wicket:id="name" type="text" maxlength="10"
         name="" id="name" class="text size6"/></td>
     </tr><tr>
      <td><label wicket:id="page.description" for="description"/></td>
      <td><input wicket:id="description" type="text" maxlength="10"
         name="" id="description" class="text size6"/></td>
     </tr><tr>
      <td><label wicket:id="page.price" for="price"/></td>
      <td><input wicket:id="price" type="text" maxlength="10"
         name="" id="price" class="text size6"/></td>
     </tr><tr>
      <td><label wicket:id="page.supplies" for="supplies"/></td>
      <td><input wicket:id="supplies" type="text" maxlength="10"
         name="" id="supplies" class="text size6"/></td>
     </tr><tr>
      <td colspan="2">
       <table><tr><wicket:container wicket:id="images">
         <td><img wicket:id = "image" width="220" height="220"/></td>
        </wicket:container></tr></table>
       <div wicket:id="pager" class="box-section"></div>
      </td>
     </tr><tr>
      <td colspane="2"><p class="actions">
        <a href="#" class="back" wicket:id="backButton">
         <span>Back</span></a></p></td>
     </tr></table></form></wicket:extend></body>
</html>
```

Download all listings in this issue as text

PHOTOGRAPHY BY BOB ADLER

OpenJDK

Get Collaborative

Oracle's **Donald Smith** discusses the OpenJDK community, the place to collaborate on Java SE. **BY JANICE J. HEISS**

Donald Smith has spent more than a decade serving the Java community. After working as an engineer, consultant, and product manager for several information technology companies, he joined Oracle in 2002, where he served until 2005 as director of technology evangelism. He next moved to the nonprofit Eclipse Foundation, where he was director of ecosystem development from 2006 to 2011. Smith currently works for Oracle Canada ULC on the product management team for Java SE, where he is primarily responsible for the OpenJDK community.

When he returned to Oracle in May of 2011, Smith remarked in his blog about his new job, "The team I'm on has one simple mandate—keep Java the number one computing platform in the world."

Java Magazine: What is OpenJDK?

Smith: Basically, OpenJDK is the place to collaborate on an open source implementation of Java SE and related projects. It's distinct from some of the other open source Java communities, some of which are hosted at Oracle and some of which are hosted elsewhere. So, for example, the GlassFish community is a place to collaborate on Java EE. Eclipse is a place to collaborate on Java tools and runtimes at the Eclipse Foundation. The NetBeans community is a place to collaborate on Java tools at Oracle, and so on. There are lots of different communities in the Java ecosystem, and OpenJDK is the one for doing Java SE work.

Java Magazine: Give us some perspective on the history of OpenJDK, which goes back to 2006 when Sun initially open sourced Java.

Smith: It was first announced in November 2006, and in 2007 the actual project was created, so people could access the code. From 2007 to 2009, a lot of formation took place as more committers joined. And then, in 2009, Oracle began the process of acquiring Sun, which officially closed in January 2010, and that gave Oracle a chance to reveal its commitment to OpenJDK and ongoing plans.

The difference between
OpenJDK and Oracle JDK

Want to collaborate on an open source version of Java SE? Head to OpenJDK. Both individuals and companies (Apple, IBM, Oracle, Red Hat, and more) contribute code to OpenJDK. Oracle uses OpenJDK as the code base for Oracle JDK, and other organizations—particularly Linux distributions—produce builds of the Java platform based on OpenJDK sources for their distributions. Oracle JDK is a specific implementation of the JDK that is produced by Oracle. Although the code bases are very close, Oracle JDK has some minor differences. For example, Oracle JDK includes closed-source third-party components like a graphics rasterizer, third-party fonts, and additional documentation. Over time, Oracle intends to open source as much of Oracle JDK to OpenJDK as it can, except commercial features (such as Oracle JRockit Mission Control). Most of this open sourcing work involves replacing closed-source third-party components with open source alternatives or negotiating licensing agreements with third parties.

Why use one over the other?

From a feature perspective, there is not much difference between Oracle JDK and an OpenJDK build. Oracle publishes up-to-date and fully tested JDK binaries for popular development and server platforms. For most end users and developers, this is the most convenient way to get Java. If you are an enterprise user and need long-term support, or you want the most-predictable binary distribution for an ISV application, Oracle JDK is the better choice of the two. If you want to build, create, and tweak your own JDK, perhaps to address a niche performance variable or to support a specialized platform, you will want to use OpenJDK as your code base.

—Tori Wieldt

Unfortunately, I think some people assumed the worst during the quiet period of 2009 during the acquisition, which as a bystander at the time seemed to take forever. But as soon as they could, Oracle made it clear that we were serious about OpenJDK. We wanted it to be the place to do Java SE development, first and foremost, and we wanted a lot more diversity, which means having a lot of the key Java stakeholders participate.

So Red Hat has been participating for several years, and companies like IBM and SAP have joined as well. It was also great to have Apple announce with us that they were going to contribute some code with the Mac OS X port. And it's not all about corporate participation—a lot of individuals participate in OpenJDK as well and they are very important to the project. So I think a lot of people have been happy to see that Oracle is serious about doing Java SE development in the open. SUSE, a business unit of the Attachmate Group, has recently joined the OpenJDK project.

In 2011, we started to get some momentum going once the organizational structure was more established. We got the governance board up and running, and adopted a new governance model, which describes the development process. And we also launched Java SE 7 in 2011. The reference implementation for Java SE 7 was based on OpenJDK sources exclusively, so OpenJDK 7 became the reference implementation for Java.

When OpenJDK was first started (in JDK 6), much of the development was done internally inside Sun and pushed out somewhat haphazardly. The good news is that both the JDK 8 project and the JDK 7 updates project started in the open, which is an important step as we find our stride.

Also in 2011, we showed some more momentum by having Twitter, Azul, and a few others announce their participation.

THE GOOD NEWS

Diversity is important for open source projects. The good news is that we have about 350 developers in the OpenJDK community, and about 80 of those are not employed by Oracle. This is a good start.

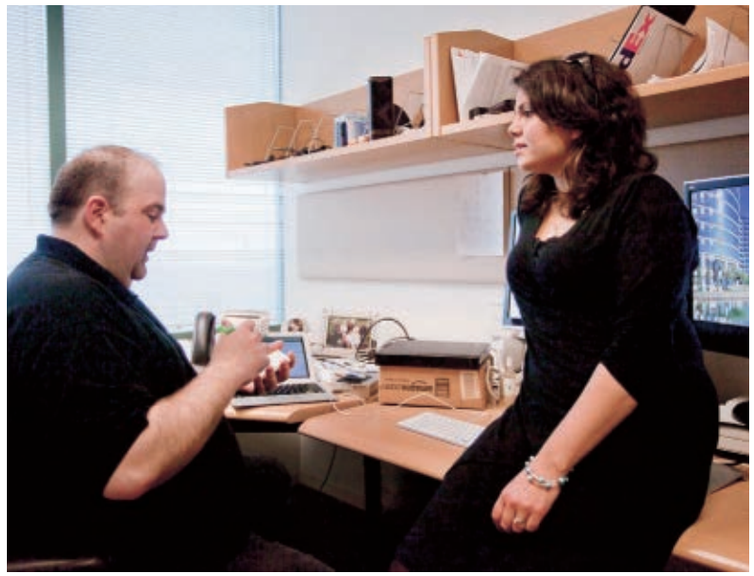
Java Magazine: Can you give us some concrete numbers that shed light on OpenJDK diversity?

Smith: First, let's agree that diversity is important for open source projects. One of the lessons I learned at Eclipse is that diversity is an important metric for the quality and longevity of projects. Optimally, you want to have a diverse mix of types and sizes of organizations and some key individuals participating in an open source project. It's also a plus to have diversity based on geography, experience, and other factors.

For OpenJDK, the initial code base and contribution came from Sun and then Oracle so we started from a position of low diversity, which is typical of many open source communities that start with a large initial contribution. But we want to keep pushing in the direction of more diversity. We can debate how fast and how far we move toward this, but pretty much everyone in the community agrees that we seek diversity.

The good news, based upon the census taken when the bylaws were approved in mid-2011, is that we have about 350 developers in the OpenJDK community, and about 80 of those are not employed by Oracle. This is a good start. We can also look at diversity in terms of e-mail traffic—about 40 percent of the e-mail traffic, according to MarkMail, is from non-Oracle employees.

Java Magazine: Can you talk about the OpenJDK community Technology Compatibility Kit [TCK] for Java SE 7?



Left to right: Oracle's Donald Smith, director of product management, and Dalibor Topic, principal product manager, OpenJDK, make plans for an OpenJDK community event. Smith and Cecilia Borg, OpenJDK onboarding program manager, talk about recognizing achievements in the OpenJDK community.

Smith: The OpenJDK community has access—through a TCK license agreement specifically for the OpenJDK community—to a compatibility test kit to validate that their builds of OpenJDK are compatible with the Java SE specification. This license is free of charge, but you have to sign the agreement, and we have to process it, which can take a few weeks. You can find the agreement on the OpenJDK Website. We're trying to keep Java compatible.

Java Magazine: What's happening now in terms of industry participation in the OpenJFX community?

Smith: We announced at 2011 JavaOne that we were going to begin the process of open sourcing JavaFX, and we've made strides with initial code contributions, and now we're looking to ramp up community participation. Several organizations are making strategic use of JavaFX, and we're hoping to see more participation in the project. We anticipate having some exciting announcements related to JavaFX at JavaOne this year.

Java Magazine: Tell us about the various groups who are a part of OpenJDK.

Smith: The notion of a *group* is codified in the OpenJDK bylaws. A group is a collection of participants interested in engaging in an open conversation about a common interest. There are, for example, groups that focus on the core libraries, the compiler, and security. One of the more topical areas right now that my team is focusing on is the quality group, where we want to make it easier for people to submit and run their own test cases and test harnesses. There is a group related to build. The challenge with both of these groups, and a number of our activities, is that we are trying to push the infrastructure along to keep up with the demands of the community. We're working on a better bug reporting system, but we also need a better build and test story so that people can more easily get their own builds up and running.

SOLID FUTURE

I believe we've established Oracle's fundamental commitment to Java through OpenJDK. We have a roadmap for releases, a Java Virtual Machine strategy for HotSpot and Oracle JRockit, and many key industry participants.

Luckily, we are getting help. The London Java Community has done a great job of pushing along information about how to do builds and providing very constructive and timely feedback on these topics. And there's a developer from SAP named Volker Simonis who has done a fantastic job at documenting his own experiences building OpenJDK on a number of platforms. There are lots of stars in the community, but we're always looking for more!

Java Magazine: OpenJDK contributors now include such companies as Apple, and as you mentioned, IBM and Twitter. It's noteworthy that Twitter joined, even though they are not a company that gets their revenue from software licensing—they're an end user of Java.

Smith: This is a very important point. When I was at Eclipse, we noticed something that academics and others have written about. There are different waves of participation in open source communities. The first wave is usually one or two organizations that have a significant code base that, for whatever reason, they're interested in making open source. They are usually ISVs [independent software vendors], or other enterprise software companies, that make their revenues directly from software licensing.

Then, after the project has proven itself, a second wave of participation occurs, consisting of companies that are software companies but don't generate their revenue directly from software licenses. So software to them is not their

Smith maps out an upcoming presentation.

product. Twitter is a great example. They build software—their biggest expense is probably software development—but they don't sell software per se. They sell a service and generate revenue from advertising and data intelligence. They're savvy with software but don't sell software licenses.

And then the third wave of participation is another degree removed from that—banks, insurance companies, maybe shipping companies, retailers, and so on. These organizations generate none of their revenue from software but still use software internally for strategic reasons.

Java Magazine: So what is the business community model that makes OpenJDK, or any open source projects for that matter, work?

Smith: The fundamentals of business suggest that there are three ways to increase the value of your company. You can increase revenue; you can lower costs; or you can increase the multiple that somebody who would want to buy your company would assign you based on your agility, your ability to grow, or whatever. Open source and participating in open source can help in all three ways.

People sometimes say that open source lowers costs because you share the development cost with other organizations. But it doesn't really work that way. Instead, different organizations focus on what they're good at. It's not so much that you lower your costs. But you end up with a richer product with greater functionality than you could do alone. You get more for your dollar because new opportunities arise. In that sense, it lowers costs because open source can often cost less to create higher-quality products. You can drive products in new directions, which helps with revenue.

Java Magazine: Describe the virtuous cycle that drives open source communities.

Smith: As with any cycle, you need a starting point. In most cases, I've noticed that you initially start out with a code base, which is the project. And that project gets adopted by end users. In the case of OpenJDK, there are two kinds of users: there are Java developers, but also the millions of end users who have Java on their desktops and servers.

With a large pool of users, whether it's a large pool of developers or end users with Java on their desktops and servers, there exists a business or personal opportunity that attracts vendors and individuals. The vendors will usually be looking for profit in the form of cash. Individuals may be looking for profit, but they may also be simply looking for the satisfaction of having helped and participated in an important software project. Or they may be looking to hone their skills or get the personal satisfaction of fixing a problem.

SPECIALISTS NEEDED

Many contributors may assume that they don't have enough expertise across the Java platform. What is really needed is deep specialization, and there are many technologies to specialize in.

So those benefits tend to attract more committers who either work for companies or for themselves, who improve the quality and robustness of the software, which in turn strengthens projects, which increases adoption and attracts more vendors, which leads to more committers, which improves the quality, and so on. That's the cycle.

Java Magazine: How can people get more involved in OpenJDK?

Smith: There are many ways to get involved. Many contributors may assume that they don't have enough expertise across the Java platform to be able to contribute, but that's often not the case. What is really needed is deep specialization, and there are many technologies to specialize in. We need help with Java SE 7 updates, new language features, Lambda, Jigsaw, the Mac OS X port, JavaFX, and so on.

If you observe any open source community for a while, you'll know that you can't just show up and expect to know all the norms and development processes and have the credibility to make immediate significant contributions. At Eclipse, we used to call this *meritocracy*, which means that those who demonstrate the ability to effect change will get that right if they want it. And it takes time and results to prove your worth. `</article>`

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

LEARN MORE

- OpenJDK
- Donald Smith's blog

JOSH MARINACCI

BIO

Java SE 7: Shaped and Translucent Windows Deep Dive

Enhance your applications in new and interesting ways.

One of the best new features of Java SE 7 is official support for shaped and transparent windows. While shaped windows have long been possible using various tricks and hacks, operating systems have started to support this feature natively in recent years. Updates to Java SE 6 exposed this native functionality through private APIs, but starting with Java SE 7, it is now an official part of Java SE.

With shaped and transparent windows, you can enhance your applications in ways that simply weren't possible before. This article takes you on a deep dive into the technology and shows you some advanced ways to enhance your applications with it. If you would like to code along with me, download the full project source code.

I will assume that you are already familiar with the shaped and translucent window concept and API. If not, Oracle has a great tutorial on the topic. The Java SE 7 API exposes three kinds of window effects: shaped windows, transparent windows, and windows with a full alpha channel. I won't cover the first two types because I have found their usefulness to be limited, and the most common use cases are already covered in the standard tutorials. Instead, I will focus on the third kind: windows with a full alpha channel.

Background

Working with a per-pixel alpha channel is more powerful than shaped and transparent windows because you can control the transparency of each pixel independently. A shaped window is simply a nonrectangular window. All visible pixels are still 100 percent opaque, but the window might be circular or some other funky shape. Translucent windows set a single transparency value for the entire window, which can be useful if you want to fade the window all at once, but it's not useful for much beyond that. The per-pixel alpha channel is where the real magic is.

In computer graphics, we define colors with four components: red, green, blue, and alpha. *Alpha* represents the transparency of the color. A computer image is just an array of pixels, in which each pixel has one or more of those four color components. The image is said to have an alpha channel if each pixel has an alpha component.

SHAPE SUPPORT

While shaped windows have long been possible using various tricks and hacks, they are now an official part of Java SE 7.

All modern desktop operating systems draw each window into an offscreen buffer before copying the buffer to the screen. The operating system uses the alpha part of each pixel to decide how to blend the pixel with the rest of the windows on the screen. By having control over the alpha channel, we can make all sorts of interesting effects, such as fuzzy edges, translucent overlays, windows with holes in them, and cool pop-down menus.

About Screen

Let's start off with something simple: an "about" screen with fuzzy edges. This could be the screen that a user sees when selecting the About menu item from the system menu of your application. First, I will create a Swing component that draws some text on top of a translucent gradient. The code is pretty straightforward Java 2-D code, which fills a rectangle with a gradient and then draws text on top, as shown in **Listing 1**.

Notice that the gradient is made of four separate colors and each color has an alpha component (the last argument to the

PHOTOGRAPH BY
CHRIS PIETSCH/GETTY IMAGES

Figure 1

Color constructor). The code in **Listing 1**, by itself, won't actually look transparent onscreen, however. The component will be partially transparent, but all you will see is the background of the window. In Swing, every window has a standard background color (usually some form of gray) on which every component is drawn. To truly make the window transparent, we need a few more things (see **Listing 2**).

The code in **Listing 2** creates the window that the component will be in. It turns off the window decorations and then sets the background to the color 0,0,0,0, which is fully transparent. This effectively turns the background of the window off, letting the desktop background or other applications shine through. Combining this with our custom component, the app looks like **Figure 1**.

The translucent window in **Figure 1** looks great, but there are a few problems. First, because we turned off the window decorations, the user can't move the window. There is no title bar to drag. Second, the user can see the window in the taskbar, even though we may not want that. It's not enough to just make the window look cool. We have to consider *how the user will actually use it*. Let's look at another example where we solve these problems.

LOTS OF OPTIONS

In the old days, we had a fairly small number of UI controls: buttons, sliders, checkboxes, scroll bars, and menus. **Today we have many kinds of custom controls.**

Keystroke Overlay

I often do demonstrations of programs onscreen. The audience can see my mouse cursor but they often don't know what keys I'm pressing, especially if I'm using menu shortcuts, which means nothing shows up onscreen when I press the keys. In these cases, it would be nice to have a window at the bottom of the screen that would show my keystrokes in a nice large font overlaid on top of the app. With a translucent window, it would be a snap.

First, we need to capture the keystrokes. I could put a keyboard listener on every text component in my entire application, but there is an easier way. Swing lets you add a listener directly to the event queue, which enables you to see all the keystroke events in one place. **Listing 3** shows a code snippet of adding the AWTEventListener.

The code in **Listing 3** adds a listener to the main event queue. It uses a mask to filter out anything that isn't a keystroke. If the keystroke event is a keyPressed event (rather than a keyReleased or keyTyped event), the code will tell the overlay window about it. Now let's move on to the overlay window.

The overlay will be a translucent, gray, round-cornered rectangle with large white text in the middle. To do this, I created an overlay component that draws everything using Java 2-D code. The keystroke event listener will set the text using the setText() method, which will trigger a repaint and draw the keystroke onscreen. See **Listing 4**.

Now let's put the overlay in a window. Again, we create a window, turn off decorations, set the background to transparent, and add the component, as shown in **Listing 5**. There's one slight difference, though; can you see it?

Instead of using JFrame, I used JWindow. JWindow is the base class of JFrame and behaves very similarly, but with one important difference. A JFrame is a full top-level window with decorations, and it will show up in the operating system's native task manager. In the case of Microsoft Windows, this means the taskbar. A JWindow is simply a window.

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

```
private static class AboutComponent extends JComponent {
    public void paintComponent(Graphics graphics) {
      Graphics2D g = (Graphics2D) graphics;

      //create a translucent gradient
      Color[] colors = new Color[]{
              new Color(0,0,0,0)
              ,new Color(0.3f,0.3f,0.3f,1f)
              ,new Color(0.3f,0.3f,0.3f,1f)
              ,new Color(0,0,0,0)};
      float[] stops = new float[]{0,0.2f,0.8f,1f};
      LinearGradientPaint paint = new LinearGradientPaint(
                  new Point(0,0),
                  new Point(500,0),
                  stops,colors);
      //fill a rect then paint with text
      g.setPaint(paint);
      g.fillRect(0, 0, 500, 200);
      g.setPaint(Color.WHITE);
      g.drawString("My Killer App", 200, 100);
    }
  }
```

Download all listings in this issue as text

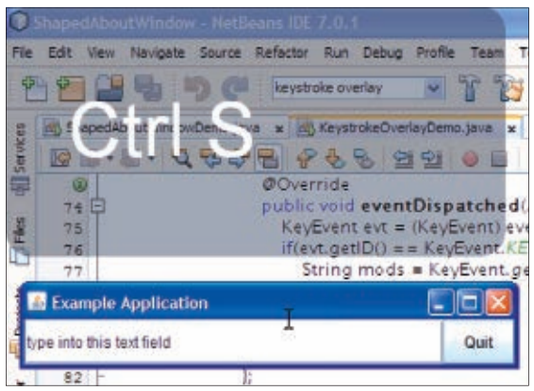


Figure 2

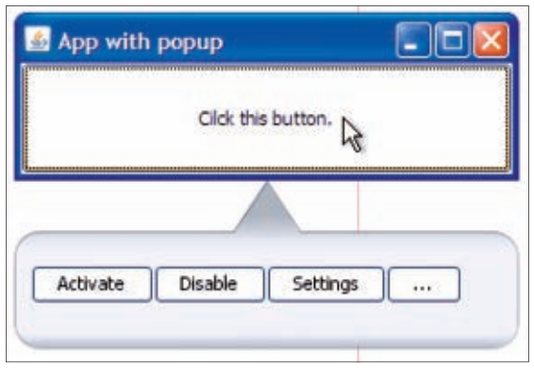


Figure 3

It doesn't represent a large part of your app, so it won't show up in the taskbar and it doesn't have decorations. On Mac OS X, it won't show up in the Expose view either. A JWindow is used for transient things, such as drop-down menus and combo boxes. Because our keyboard overlay is not a large, functional part of the app, using a JWindow is perfect.

Figure 2 shows what the final app looks like. As I type into any text field or press menu shortcut keys, the most recent keystroke appears in the overlay window.

Fancy Pop-Down Menus

In the old days of windowing interfaces, we had a fairly small number of UI controls: buttons, sliders, checkboxes, scroll bars, and menus. That was pretty much it. Today we have many, many kinds of custom controls that behave in lots of different ways.

A popular new kind of control in desktop apps such as Chrome is the pop-down menu (though every platform has its own name for this control). A pop-down menu is a small window containing several actions. It "pops down" from a button on the main interface and is used to expose more functionality to users without taking them away from the main window. Pop-down menus are transient. They appear when the user clicks the button and disappear when the user chooses an action or switches to another window. And, of course, they need to look cool. Sounds like a job for a translucent JWindow.

Figure 3 shows the goal: a pop-down menu with rounded corners containing the actions. It has a pointy tab at the top indicating where the pop-down menu came from, which ensures that the user never gets lost or wonders what the pop-down menu is referring to.

Drawing the pop-down menu itself is pretty easy. The buttons are just regular JButton objects in a JPanel wrapped in a custom container called the PopupTabPanel. This custom panel sizes itself to fit the child components, but we need some extra space to draw the cool rounded and pointy edges. To do that, I added an empty border using the Swing BorderFactory, and then I overrode paintComponent to draw the gradient effects. **Listing 6** shows the code.

Notice that I used the Java 2-D Area

LISTING 6

```
private static class PopupTabComponent extends JComponent {
    public PopupTabComponent(JComponent component) {
      this.setLayout(new BorderLayout());
      this.add(component, BorderLayout.CENTER);
      component.setBackground(new Color(0,0,0,0));
      this.setBorder(BorderFactory.createEmptyBorder(35, 10, 10, 10));
    }
    @Override
    protected void paintComponent(Graphics gfx) {
      Graphics2D g = (Graphics2D) gfx;
      g.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.
VALUE_ANTIALIAS_ON);

      Color[] colors = new Color[]{
              new Color(0.6f,.6f,.6f,0.9f)
              ,new Color(0.9f,0.9f,0.9f,1f)
              ,new Color(0.9f,0.9f,0.9f,1f)
              ,new Color(0.6f,0.6f,0.6f,0.9f)};
      float[] stops = new float[]{0.2f,0.4f,0.9f,1f};
      LinearGradientPaint paint = new LinearGradientPaint(
                  new Point(0,0), new Point(0,getHeight()),
                  stops,colors);

      //g.setPaint(new Color(255,255,255,120));
      g.setPaint(paint);
      Path2D.Double path = new Path2D.Double();
      path.moveTo(getWidth()/2, 0);
      path.lineTo(getWidth()/2+20, 30);
      path.lineTo(getWidth()/2-20, 30);
      path.closePath();
      Area area = new Area(path);
      RoundRectangle2D.Double rect = new RoundRectangle2D.
Double(0,30,getWidth()-1,getHeight()-30-1,30,30);
      area.add(new Area(rect));

      g.fill(area);
      g.setPaint(new Color(50,50,50,120));
      g.draw(area);
    }
```

Download all listings in this issue as text

Figure 4

class. This lets you merge multiple shapes together. I created the pointy tab shape by merging a triangle with a rounded rectangle. This was far easier than trying to write the Bezier curve code for the shape directly. Then I filled the shape with another nice translucent gradient and added a stroked border to make it stand out.

The pop-down menu looks great and it doesn't show up in the taskbar, because I used a JWindow—but there is still one problem. If the user chooses an action, the pop-down menu will close, but what happens if the user switches to another application or minimizes the main window? What if the user moves the window? The pop-down menu will still be there, just sort of hanging out in space unattached to anything.

To fix this, I need to add an extra listener for window events. If the window moves, we should move the pop-down menu. If the window is hidden or is no longer the active window, we should close the pop-down menu. The code in **Listing 7** adds listeners to the main frame to move or hide the pop-down menu as needed.

Remember, when creating this sort of custom window, we must always consider how the user will actually use it.

Magnifying Glass

For our final example, let's build something crazy that simply wasn't possible before: an onscreen magnifying glass. This is a window that actually has a hole in it. Anything inside the hole will be magnified. To make it look cool, we will use a fully custom window without any window decorations, which means we need another way for the user to move it around.

First, let's create the cool-looking window. I drew the window using Adobe Photoshop and saved it as a transparent PNG file. See **Figure 4**.

Again, I'll create a custom component inside of an alpha channel window. I also set the window to always be on top so that I can use it when interacting with other windows. **Listing 8** shows what the code looks like.

Now let's make the magnification actually work. To capture what's inside the smaller circle, we will use the java.awt.Robot API. This API was created mainly for doing testing and automation, but it has a very handy screen capture method that lets you specify a small area of the screen to capture. Once we have that small slice of screen, we will draw it blown up by a factor of 8 in the other circular area. **Listing 9** shows an excerpt of the full code available in the architect.trans-windows-project.zip.

Now we have a magnifying glass, but there are no window decorations, so the user can't move it. The most natural thing is to let the user click and drag anywhere on the window to move the magnifying glass. We can do this by attaching a mouse listener to the main component. Every time the user drags, we will update the position of the window and then update the screen cap-

LISTING 7 LISTING 8 / LISTING 9 / LISTING 10

```
        frame.addComponentListener(new ComponentAdapter() {
          @Override
          public void componentMoved(ComponentEvent e) {
            tab.hideTab();
          }
        });
        frame.addWindowListener(new WindowAdapter() {

          @Override
          public void windowClosed(WindowEvent e) {
            tab.hideTab();
          }

          @Override
          public void windowIconified(WindowEvent e) {
            tab.hideTab();
          }

          @Override
          public void windowDeactivated(WindowEvent e) {
            tab.hideTab();
          }
        });
        frame.add(button);
        frame.pack();
        frame.setLocationRelativeTo(null);
        frame.setVisible(true);
        frame.setSize(300, 100);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

Download all listings in this issue as text

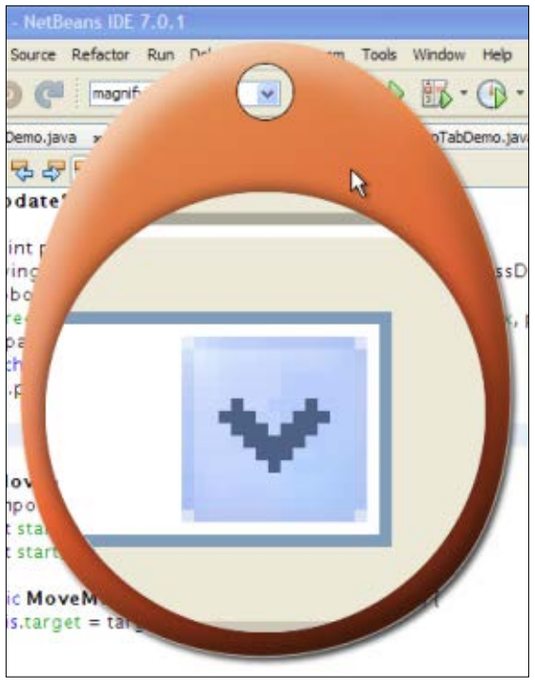


Figure 5

ture. Presto: a real magnifying glass.

In the code in **Listing 10**, be sure to note that we first transform the coordinates into screen space and then add the mouse drag offsets. It is important that you always convert into screen space to avoid any error buildup that might be caused by moving the coordinate space that the events are calculated in.

If you run this app and try clicking in the small circular area, you will see that mouse events go through the circle to the window below. This really is a window with a hole in it (see **Figure 5**), which brings up a question: How does the operating system know what the edges of the window really are? We didn't set any special API or forward mouse events down to the lower window. Well, the OS will use the alpha channel. If the user clicks a pixel that might be in the window, but the pixel is fully transparent, then the event will go through. If the pixel is fully opaque or partially transparent, then the pixel will be sent to your application.

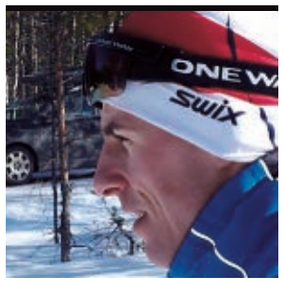
This brings up another question: Could you create a window that was 99 percent transparent with a tiny but nonzero alpha value? If you could, would the clicks be captured or go through? The exact behavior isn't defined in the Java SE 7 API spec, but testing reveals that the operating system uses a threshold. If your alpha value is small and below that threshold, but not quite zero, the OS will treat the value as zero and send the mouse click through. This prevents an evil program from covering the entire screen with a nearly transparent window to maliciously capture mouse clicks or otherwise fool the user. My testing reveals that a value of 10, or 2.5 percent, will work. `</article>`

POPULAR CONTROL

The **pop-down menu** is a popular new kind of control in desktop apps. This small window exposes users to more functionality.

LEARN MORE

- Java SE 7 API specification



KNUT HATLEN

BIO

What's New in Java DB in JDK 7

Improvements now make it even easier to start using databases and SQL in your apps.

Java DB is a relational database written purely in the Java programming language. It is based on and is compatible with the open source Apache Derby database.

Java DB has been available as part of the JDK since JDK 6, and it can be found in the db subdirectory of the JDK installation. Its presence in the JDK makes it easy for Java developers to start using databases and SQL in their applications.

Because Java DB adheres closely to the JDBC and SQL standards, it is upwardly compatible with many enterprise database systems. You can use Java DB as a starting point for developing data-rich applications, and then switch to an enterprise-caliber database later.

The version of Java DB in JDK 6 was based on Apache Derby 10.2. The version in JDK 7 is based on Apache Derby 10.8, which has many improvements and new features compared to the old version. This article surveys some of these improvements and is based on a presentation that Rick Hillegas, a member of the Java Class Libraries team at Oracle, gave at JavaOne 2011.

GET STARTED

You can use Java DB as a **starting point for developing data-rich applications**, and then switch to an enterprise-caliber database later.

Better Integration with Java Code

Java DB has always allowed you to write user-defined routines in the Java language and use them in your SQL statements. In JDK 7, Java DB has two new features that enable you to take even more advantage of the Java platform inside the database: *user-defined types* and *table functions*.

User-defined types. User-defined types allow you to define new datatypes that correspond to Java classes or interfaces and use those types just like any of the built-in datatypes, as columns in a table or as parameters in functions or procedures. The only requirement is that the Java class must implement the java.io.Serializable interface.

The user-defined types can be defined both for your own application classes and for classes that exist in the standard class library of the Java runtime environment. **Listing 1** shows an example of how to define and use a type that allows you to store java.util.List objects in a table.

Table functions. A table function is a variant of user-defined functions. In contrast to an ordinary user-defined function, which returns a scalar value, a table function returns a tabular data set and can be used in the FROM clause of a SELECT statement. This allows you to query heterogeneous datasources using SQL.

Table functions are implemented as static Java methods that return instances of java.sql.ResultSet. Java DB provides an abstract class, org.apache.derby.vti.VTITemplate, with default implementations for most of the ResultSet methods, which makes it easier to write a custom ResultSet class.

Listing 2 shows an example of how to use VTITemplate to write your own table function. That table function reads a text file and returns a table with one row per line in the file, where each row has two columns: the line number and the text found on the line.

The following are the most important parts of the table function class:

- The public static readFile() method that creates a new instance
- The next() method that reads the next line of the file
- The getInt() and getString() methods that return the column values for the current row

To wire the table function into your database, you need to declare it using a CREATE FUNCTION statement. **Listing 3** shows how to do that in ij, the Java DB interactive scripting tool, and it shows a sample query that uses the table function.

Other Usability Features

Generated columns. A *generated column* is a column whose value is defined as an expression. The

expression may refer to other columns in the same table, and the value of a generated column is automatically updated if any of the columns on which it depends are modified.

For example, if you have a table that contains people's birthdays, and you frequently need to fetch all people who have birthdays in the current month, you could define a BIRTHMONTH column that is automatically generated based on the BIRTHDAY column:

```
CREATE TABLE PERSONS(
  NAME VARCHAR(50),
  BIRTHDAY DATE,
  BIRTHMONTH GENERATED ALWAYS
    AS (MONTH(BIRTHDAY)))
```

In order to speed up the queries, you can also create an index on the generated column:

```
CREATE INDEX MONTH_IDX ON
PERSONS(BIRTHMONTH)
```

And as new people are added to the table, or existing records are updated, the database automatically updates both the generated column and the index. This way, generated columns can be used as a substitute for expression indexes, which are found in some other database systems.

Sequence generators. The latest version of Java DB also supports *sequence generators*. These are SQL objects that generate monotonically increasing identifiers. They are created by issuing an SQL statement such as CREATE SEQUENCE S, and then values can be retrieved from the sequence by using the expression NEXT VALUE FOR S, for example:

```
CINSERT INTO CUSTOMERS(ID,NAME)
  VALUES (NEXT VALUE FOR S, 'Joe')
```

Limiting results (OFFSET/FETCH NEXT). One of the most frequently requested features for Java DB has been a way to limit the number of rows returned by a query. The latest version of Java DB now supports the OFFSET/FETCH NEXT syntax that was introduced in SQL:2008.

The OFFSET clause indicates how many rows to skip at the beginning of the result, and the FETCH NEXT clause tells the maximum number of rows to fetch. These clauses can be used separately or in combination.

For example, to select the three highest scores from a table, you would use FETCH NEXT like this:

```
SELECT NAME, SCORE FROM RESULTS
  ORDER BY SCORE DESC
  FETCH NEXT 3 ROWS ONLY
```

And to select the scores ranked from 11 to 20, you would combine OFFSET and FETCH NEXT like this:

```
SELECT NAME, SCORE FROM RESULTS
  ORDER BY SCORE DESC
  OFFSET 10 ROWS
  FETCH NEXT 10 ROWS ONLY
```

Language-based ordering. By default, Java DB orders character strings by doing a character-by-character comparison of

LISTING 1 LISTING 2 / LISTING 3

```
// Define a new type for java.util.List
stmt.execute("CREATE TYPE JAVA_UTIL_LIST " +
        "EXTERNAL NAME 'java.util.List' " +
        "LANGUAGE JAVA");

// Create a table that holds java.util.List objects
stmt.execute("CREATE TABLE T(LST JAVA_UTIL_LIST)");

// Insert lists into the table
try (PreparedStatement ps = conn.prepareStatement(
     "INSERT INTO T(LST) VALUES (?)")) {
   ps.setObject(1, Arrays.asList("First element", "Second element"));
   ps.executeUpdate();
   ps.setObject(1, Arrays.asList("A", "list", "with", "five", "elements"));
   ps.executeUpdate();
}

// Retrieve the lists from the database
try (ResultSet rs = stmt.executeQuery("SELECT LST FROM T")) {
   while (rs.next()) {
      List list = (List) rs.getObject(1);
      System.out.println("Size of list: " + list.size());
   }
}
```

Download all listings in this issue as text

the Unicode code points. Although this is fine in many cases, it does not necessarily result in the same ordering as you would expect in a phone book or dictionary. For example, an uppercase letter is not sorted close to its corresponding lowercase variant, and punctuation characters are considered to be significant.

In JDK 7, Java DB can also order character strings by using the collation rules defined for specific locales, so-called *language-based ordering*. These rules typically resemble more closely what you would find in a phone book.

Language-based ordering has to be enabled using connection attributes when the database is created. To create a database that obeys the collation rules of the en_US locale, you can use the following URL:

```
jdbc:derby:db;create=true;
territory=en_US;
collation=TERRITORY_BASED
```

Table 1 shows how a set of strings would be ordered by Java DB with default ordering (on the left) and with language-based ordering (on the right). Notice that all the names that start with capital letters are sorted before the names that start with lowercase letters in the default ordering, whereas they are interleaved in between the other words, in alphabetical order, with the language-based ordering.

DEFAULT ORDERING	LANGUAGE-BASED ORDERING
VANDERBILT	DA VINCI
WOOLF	VAN BEETHOVEN
DA VINCI	VANDERBILT
VAN BEETHOVEN	VAN GOGH
VAN GOGH	VON GOETHE
VON GOETHE	WOOLF

Table 1

In-memory database. Java DB has also introduced an in-memory storage back end that stores the entire database in main memory and doesn't leave any data on the disk. This, of course, means that the database disappears as soon as the application stops, so you would use this mode only if the database contains transient data that you can afford to lose.

For example, if you are writing unit tests for your application, it is probably more important that the tests run quickly and that it is easy to clean up after the tests. With the in-memory storage back end, many database operations (such as database creation, insertions, and updates) are much faster, because they don't need to access the disk. Also, there is no need to clean up and delete the database files after the tests have been completed, because the database goes away when the application terminates.

Enabling the in-memory back end is quite simple. You just add the `memory` subprotocol to the JDBC connection URL. No other changes should need to be made to your application. Connecting to a database using the following URL would create a new in-memory database:

```
jdbc:derby:memory:MyDB;create=true
```

More or less, everything you can do with an ordinary database can be done with an in-memory database, including taking a backup of it and restoring it. This allows you to dump the database to disk before you shut down your application, reload it into memory the next time you start the application, or even turn it into an ordinary on-disk database.

Asynchronous replication for high availability. *Asynchronous replication* can be used to minimize downtime in case the database server crashes for some reason. When running in this mode, all modifications in the database are replicated on a copy of the database on another server. If the database server goes down, the application can tell the copy to take over and continue as if nothing has happened.

The Java DB Server and Administration Guide contains the details on how to set up replicated databases.

Performance Enhancements

Many performance enhancements have gone into Java DB between JDK 6 and JDK 7. The code paths of many common operations have been optimized so that fewer CPU cycles are spent when performing queries. Also, some performance-critical modules have been rewritten to use collection classes from the `java.util.concurrent` package,

which has greatly improved the scalability for multithreaded use of Java DB.

Another area of improvement is large objects (BLOBs and CLOBs). Whereas the old version of Java DB tended to materialize the large objects in memory when reading or manipulating them using the network client, the new version streams the objects as much as possible. This reduces the memory requirements on the clients and makes accessing large objects faster in many cases.

On the optimizer side, Java DB now automatically updates index cardinality statistics. These statistics are used by the optimizer to decide which query execution plan is more likely to yield the best performance. In older versions, the statistics had to be updated manually, and outdated statistics were a frequent source of poor performance. The new version of Java DB is able to detect that the statistics are out of date and automatically schedule a background task that updates the statistics.

IT'S AUTOMATIC
The optimizer uses cardinality statistics, which are **automatically updated**, to decide which query execution plan will yield the best performance.

Security Enhancements

SSL/TLS network encryption. For safe communication between a Java DB server and its clients over the network, support for network encryption using Secure Sockets Layer (SSL) and Transport Layer Security (TLS) has been added. To enable it, you first need to create certificates for the server and (optionally) the clients.

Once the certificates are generated, SSL/TLS encryption can be switched on by starting the Java DB network server with the `-ssl` command-line option and adding an `ssl` connection attribute to the connection URL on the clients.

Improved management of privileges. Java DB has supported SQL authorization for a long time, so you can use GRANT and REVOKE statements to specify which users have privileges to certain operations. The version in JDK 7 has two new features that make it easier to manage privileges: *SQL roles* and *definer's rights*.

SQL roles allow you to grant sets of privileges to users. So if, for example, some of your database users need to be able to read and make updates to the CUSTOMERS table and also execute a stored procedure called ADD_CUSTOMER, you could create a role that holds all those privileges:

```
CREATE ROLE CUSTOMER_ADMIN
GRANT SELECT ON CUSTOMERS
  TO CUSTOMER_ADMIN
GRANT UPDATE ON CUSTOMERS
  TO CUSTOMER_ADMIN
GRANT EXECUTE
  ON PROCEDURE ADD_CUSTOMER
  TO CUSTOMER_ADMIN
```

Instead of granting each individual privilege to every user who needs rights to modify the customer data, the newly created role can be granted instead, which implicitly grants the privileges to users:

```
GRANT CUSTOMER_ADMIN TO JOE
GRANT CUSTOMER_ADMIN TO ANN
```

Later, if Joe doesn't need these privileges anymore, they can be revoked with a single REVOKE statement:

```
REVOKE CUSTOMER_ADMIN FROM JOE
```

Definer's rights affect how privileges are checked in stored procedures. The default (called *invoker's rights*) is to require that the user invoking the procedure has privileges to perform every database operation in the procedure. With definer's rights, the user who invokes the procedure needs only the privilege to execute it; the procedure runs with the privileges of the user who defined it.

This makes it possible to delegate more fine-grained privileges. For example, instead of granting a user the privilege to insert rows into a table, which would mean that the user is free to insert anything into the table, a stored procedure could be defined that inserts a row, possibly after doing some sanity checks first. The procedure needs to be defined by a power user who has the required privilege to insert rows into the table, but the users who call the procedure need only an execute privilege.

To define a procedure that's invoked with definer's rights, an EXTERNAL SECURITY clause must be added to the CREATE FUNCTION or CREATE PROCEDURE statement, for example:

```
CREATE PROCEDURE ADD_CUSTOMER()
  LANGUAGE JAVA
  PARAMETER STYLE JAVA
  EXTERNAL NAME
    'DbProcs.addCustomer'
  EXTERNAL SECURITY DEFINER
```

Conclusion

We have scratched the surface and taken a quick look at some of the improvements in Java DB, but there are many more. Now, download JDK 7 and try it out for yourself! For more details about the features, refer to the Java DB documentation, or join the Apache Derby user community. `</article>`

LEARN MORE

- "Java DB Table Functions" white paper (PDF)
- "Java DB Security" white paper (PDF)

RAOUL-GABRIEL URMA AND **JANINA** VOIGT

BIO

Using the OpenJDK to Investigate Covariance in Java

Generics and wildcards provide a type-safe alternative to covariant arrays.

Some programming language features, even those that are well established, can be controversial. An example of such a controversial feature is *covariance of arrays* in Java, which has long been seen as a weakness in the language's type system. Covariant use of arrays causes type errors that cannot be identified by the Java compiler and result in runtime exceptions when the program is executed.

This article first explains the meaning of covariance and illustrates the typing problems it causes in the context of arrays in Java. It then presents the results of an empirical study we conducted to ascertain how often arrays are used covariantly by Java developers in practice. This empirical study fills a gap in the current discussion around covariance by providing data about the use of the feature, and we hope that it can help inform the design of future programming languages.

Covariance in Java

Java is statically typed, and its compiler will detect most typing errors before a program is executed. This is useful because it means that a number of mistakes in a program can be found before execution and testing.

However, there are some type errors in Java, for example, those caused by typecasting, that cannot be detected at compile time and, thus, can cause runtime exceptions. Consider the following piece of code:

```
Banana[] bananas = new Banana[5];
Fruit[] fruit = bananas;
fruits[0] = new Apple();
// Previous line causes runtime
// exception
peelBanana(bananas[0]);
// Apple??
```

The example above is accepted by the Java compiler (assuming that Apple and Banana are both subclasses of Fruit) but causes a runtime exception when executed.

The problem is caused by Line 2, where we take an array of Bananas and assign it to an array of Fruit. Since a Banana is a subtype of Fruit, it seems intuitive to expect that Banana[] is also a subtype of Fruit[], and this is indeed confirmed by the behavior of the compiler. Unfortunately, when we try to put an Apple into our Fruit[] we get a runtime exception. Although statically (and as far as the compiler is concerned) the array has type Fruit[], its runtime type is Banana[] and, thus, we cannot use it to store an Apple.

WEAK LINK

Covariance of arrays has long been seen as a weakness in Java's type system.

The assignment statement on Line 2 is called a *covariant assignment*. *Covariance* means using an instance of a subclass where an object of the superclass is expected, for example, using a Banana where a Fruit is required. *Contravariance* refers to the opposite situation, for example, using a Fruit in place of a Banana.

In this work, we focus particularly on covariance in the context of collections and arrays. Arrays in Java are *covariant*; thus, a Banana[] can be used where a Fruit[] is required. This, however, compromises Java's type safety and leads to potential runtime type errors, as illustrated above. As a result, runtime type checks must be performed for *every* assignment to *any* array (except arrays of primitives) to ensure that the value being written is compatible with the array type.

Despite not being type safe, covariant arrays were included in Java to allow an operation to be applied to arrays of any type. For example, consider writing a find method that searches through an array for a specific object. We want such a method to work for

any type of array, rather than writing one method for String[], one for Person[], and so on. Covariant arrays allow us to write the following:

```
public static boolean find(Object
item, Object[] arr) { ... }
```

Of course, since Java SE 5, we have a better solution. Using generics, the method can be written as follows:

```
public static <T> boolean find(T
item, T[] arr) { ... }
```

This solution is not only guaranteed to be type safe (that is, it will not cause runtime exceptions because of a problem with the array's type), but it has the additional advantage of encoding the constraint such that the type of the item to find and the type of the array must be compatible. However, this solution works only in the presence of generics. Before Java SE 5, using covariant arrays, as shown above, was the only option when implementing a method like this.

Unlike arrays, generics in Java are *invariant*. This protects us from the runtime error in our earlier example involving arrays. For example, the following code will not compile:

```
ArrayList<Fruit> f = new
ArrayList<Banana>();
```

While this code is type safe, it somewhat limits us in terms of what we can express. Sometimes we want to be able to treat all lists of Fruit in the same way, regardless of whether they are lists of Bananas or Apples. As long as we do not modify the lists, this should work without causing type errors.

For this reason, Java supports wildcards for generics, which allow us to specify a bound to the instance of a generic. In practice, it can be used to support a form of restricted covariance shown in the following code:

```
ArrayList<? extends Fruit> f = new
ArrayList<Banana>();
f.add(new Banana());
// Prev. line causes compile error
```

In Java, when we use generics with wildcards, as shown above, we then cannot make modifications. For example, in the code above, we can assign an ArrayList<Banana> to the variable f of type ArrayList<? extends Fruit>, but we cannot then add any new items to f.

Covariance in Other Languages

Most languages support some form of *variance* of types. Scala, for example, allows developers to annotate a parametric type to indicate whether it should be covariant, contravariant, or invariant (that is, neither covariant nor contravariant). This is called *definition-site variance*.

In the following example, we annotate the parametric type CoBowl so that it can be used covariantly (this is indicated by the + annotation). Assigning a CoBowl<Banana> to CoBowl<Fruit> is, therefore, allowed. The parametric Bowl type, on the other hand, has no

LISTING 1

```
public class Main
{
        public static void main(String[] args)
        {
                Object[] str = new String[10];
        }
        public static Object[] test2()
        {
                return new String[10];
        }
}
```

Download all listings in this issue as text

such annotation and is, thus, invariant. Therefore, a Bowl<Banana> cannot be assigned to a Bowl<Fruit>:

```
Bowl[T]{}    // invariant
CoBowl[+T]{ } // covariant as
// indicated by the +
val bowl : Bowl[Fruit] = new Bowl
[Banana]; // compile error
val coBowl : CoBowl[Fruit] = new
CoBowl[Banana]; // correct
```

In order to avoid type errors like the one in our original example, the mutable collections API in Scala was designed to be invariant. However, the immutable collections API can safely be used covariantly.

Some programming languages, such as Google Dart, simply allow covariance in collections in order to avoid cluttering the type system with additional complexity, despite the fact that this can potentially cause problems at runtime.

USING WILDCARDS

Java supports **wildcards for generics**, which allow us to specify a bound to the instance of a generic.

Empirical Study of Covariant Arrays

Although the type problems caused by covariant arrays in Java are well known, there has been no study showing how widely this problematic feature is used by developers in practice. We conducted an empirical study on the use of covariant arrays, aiming to determine how frequently and in which contexts they are used.

We analyzed 64 open source programs from the Qualitas Corpus, ranging from games to system software and libraries. Although the Qualitas Corpus contains 106 programs, we selected only a subset, excluding some of the programs that needed to be fixed in order to compile. A number of

programs contained missing dependencies or referenced old repositories. Our final selection includes software written both before and after the introduction of generics and comprises more than 5 million lines of code.

We customized the Java compiler that is freely available as part of the OpenJDK, modifying the type checker to make arrays invariant. In this way, we could simply use the compiler to identify any uses of covariant arrays in our code corpus, because these uses would be reported as compile errors.

This required changing the method `isSubtypeUnchecked(Type t, Type s, Warner warn)`, which is called by the compiler to check whether two types are subtypes (that is, whether or not one type is substitutable for the other). Whenever this method was called with array types, we required the types of the two arrays to be identical:

```
if (t.tag == ARRAY && s.tag ==
ARRAY) {
  return isSameType(elemtype(t),
 elemtype(s));
}
```

As a result, uses of covariant arrays are reported as a compile error. For example, the code in **Listing 1** returns the following two errors:

```
Main.java:5:incompatible types
found:java.lang.String[]
required:&java.lang.Object[]
  Object[]=new String[10];
Main.java:10:incompatible types
found:java.lang.String[]
required:java.lang.Object[]
  return new String[10];
```

Results of the Study

We ran the modified Java compiler over our corpus and discovered 324 uses of covariant arrays in 9 out of 64 programs. This is a surprisingly low number considering that our corpus exceeds 5 million lines of code, which equates to one covariant array use for every 15,000 lines of code. This indicates that covariant arrays are used very infrequently.

Figure 1 shows the contexts in which covariant use of arrays occurred. The vast majority of these uses (89 percent) occurred when a method or constructor was called with a subtype of the required array parameter type. For example, if we called our `find(Object item, Object[] arr)` method from above with an argument of type `String[]`, this would be classified as a covariant method call.

The remaining uses of covariance happened either in `assignment` statements (9 percent) or in `return` statements (2 percent) where a method returned a subtype array of its declared return type.

We further analyzed the covariant method and constructor calls to see which methods and constructors were frequently called covariantly; the results are shown in **Figure 2**. We found that 84 percent of these calls went to methods and constructors in the Java library, with only 16 percent of the covariant calls going to user-defined operations. This shows that covariant arrays are used mainly in calls to the Java library; in fact, covariant calls to library methods account for almost 75 percent of all uses of covariant arrays in our corpus.

Looking more closely at the covariant method calls to Java libraries, we found that a small number of library methods accounted for the majority of covariant method calls, as shown in **Figure 3**. The `toArray` methods of `Vector` and `List` caused 146 covariant calls in our corpus, almost half of all uses of covariant arrays. These methods take an `Object[]` parameter and return an array containing all of the elements in this list in proper sequence; the runtime type of the returned array is that of the specified array. The various methods of `JOptionPane`, which take a parameter of type `Object[]` (for example, `showInputDialog`), were also called covariantly 34 times.

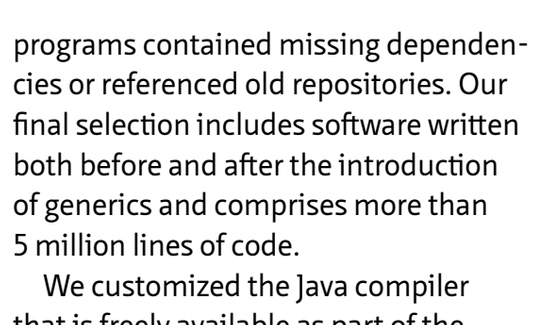


Figure 1

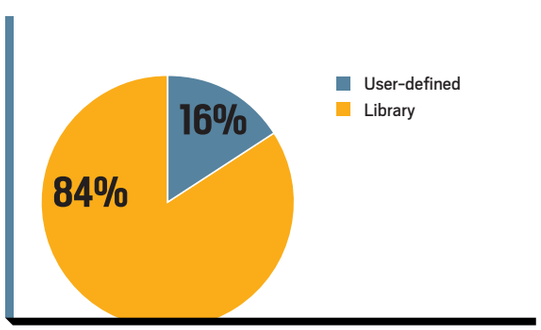


Figure 2

Conclusion

We conducted an empirical study to investigate the use of covariant arrays in real-world Java programs. Our corpus was composed of 64 open source programs from the Qualitas Corpus, ranging from games to system software and libraries. As we have shown with the help of an example, covariant arrays in Java can lead to type errors that are not caught by the compiler but cause runtime exceptions.

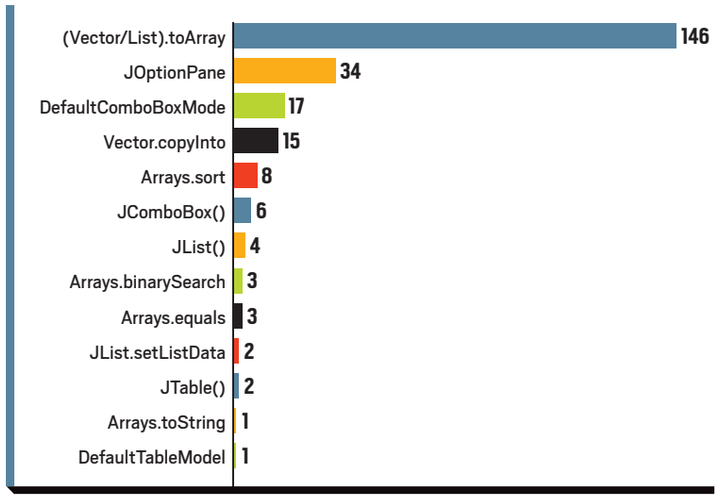


Figure 3

In our empirical study, we found relatively few uses of covariance given the large size of our corpus: 324 covariant array uses in more than 5 million lines of code. More importantly, almost 75 percent of these uses were caused by calls to Java library methods. This demonstrates that covariant arrays are used very infrequently by Java developers in practice.

Since Java SE 5, generics and wildcards provide a type-safe alternative to covariant arrays. All of the uses of covariant arrays we found in our corpus could be transformed and made completely type safe using these Java features.

The existence of generics as a better and safer solution and the infrequent use of covariant arrays in Java lead us to suggest that covariant arrays should not be included in future programming languages and should be avoided whenever possible in Java development.

We intend to extend our work by investigating the use of wildcards in Java programs. Wildcards and generics provide a better alternative to covariant arrays, and we are interested to see how frequently they are used in practice. `</article>`

We would like to thank Alan Mycroft and Luke Church for their encouragement and useful advice and suggestions during our work. We would like to thank our wider research group, the Cambridge Programming Research Group, as well as attendees of our talk at the FOSDEM conference for valuable feedback and suggestions. Further, Raoul-Gabriel Urma wishes to thank Qualcomm and Janina Voigt wishes to thank Rutherford Foundation of the Royal Society of New Zealand for supporting this work through their studentships.

LEARN MORE

- *Types and Programming Languages* by Benjamin Pierce (MIT Press, 2002)

BEN EVANS AND
PETER LAWREY

BIO

Part 1

Introduction to JIT Compilation in Java HotSpot VM

Use the PrintCompilation switch to observe the effects of Java HotSpot VM compiling methods during runs.

This article is the first article in a two-part series about Java HotSpot VM and just-in-time (JIT) compilation.

Java HotSpot VM is the VM that Oracle acquired with the Sun acquisition, and it is the VM that forms the basis of both the Java Virtual Machine (JVM) and the open source OpenJDK. Like all VMs, Java HotSpot VM's role is to provide an operating environment for bytecode. In practice, there are three major functions that need to be performed:

- Executing the instructions and computations that are requested by methods
- Locating, loading, and verifying new types (that is, class loading)
- Managing memory on behalf of application code

BEST BET

Java HotSpot VM works best when it can accumulate enough statistics to make intelligent decisions about what to compile.

The last two functions are huge topics in their own right, so in this article we will focus purely on the execution of code.

JIT Compilation

Java HotSpot VM is a *mixed-mode VM*, which means that it starts off interpreting the bytecode, but it can (on a method-by-method basis) compile code into native machine instructions for faster execution.

By passing the switch `-XX:+PrintCompilation`, you can see entries in the log file that show each method as it is compiled.

This compilation takes place at runtime—after the method has already been run a number of times. By waiting until the method is actually being used, Java HotSpot VM can make sophisticated decisions about how to optimize the code as it compiles the code.

If you're curious about how much difference the JIT makes, you can turn it off using `-Djava.compiler=none` and then look at the difference in your benchmarks.

Java HotSpot VM is capable of running in two separate modes: client or server. You can choose the mode by specifying the `-client` or `-server` switch to the JVM on startup. (This must be the first switch provided on the command line.) Each mode has different situations in which it is usually preferred. In this article, we'll be concerned only with the server mode.

The major difference between the two modes is that the server mode makes more-aggressive optimizations—based on assumptions that might not always hold. These optimizations are always protected with a simple *guard condition* to check whether the assumption is correct. If, for any reason, an assumption is not valid, Java HotSpot VM reverts the optimization and drops back to interpreted mode. This behavior means that Java HotSpot VM will never do the wrong thing due to an incorrect optimization assumption; it always checks the optimization first.

In server mode, by default, Java HotSpot VM runs a method in interpreted mode 10,000 times before compiling it. You can adjust this value by using the `CompileThreshold` switch. For example, passing `-XX:CompileThreshold=5000` causes Java HotSpot VM to run methods only half as many times before compiling.

It can be tempting for new users to reduce the compile threshold to a very low value. However, you should resist this temptation,

BEN EVANS' PHOTOGRAPH BY JOHN BLYTHE

because it can reduce performance by spending time compiling methods that the VM doesn't run enough to recover the cost of compilation.

Java HotSpot VM works best when it can accumulate enough statistics to make intelligent decisions about what to compile. If you reduce the compile threshold, Java HotSpot VM might spend a lot of time compiling methods that are not very relevant to the code paths that are run the most. Some optimizations are performed only when enough statistics have been collected, so the code might not be as optimal as it could be if you reduce the compile threshold.

On the other hand, many developers want to achieve the better performance of the compiled mode as soon as possible for their important methods.

One standard way of solving this problem is to have a warm-up harness that exercises the code enough to force compilation (by sending test traffic into the system) after the process has started. For systems such as order management or trading systems, it's important to make sure that the warm-up harness doesn't generate real orders.

Java HotSpot VM provides a number of switches to increase the amount of information logged about JIT compilation. The most common is PrintCompilation—which we already met—but there are several others.

We're going to use PrintCompilation to observe the effects of Java HotSpot VM compiling methods during runs. First, however, we need to say a few words about the System.nanoTime() method for timing.

Timer Methods

Java gives us access to two main time values: currentTimeMillis() and nanoTime(). The former corresponds fairly closely to the time that we observe in the physical world (so-called *wall clock time*). Its resolution is enough for most purposes but not for low-latency applications.

The higher-resolution alternative is the nanosecond timer. This timer measures time in incredibly short intervals. One nanosecond is the time it takes light to move about 20 centimeters in a fiber-optic cable. By contrast, light takes around 27.5 milliseconds to travel between London and New York through fiber-optic cables.

Due to the very high resolution of nanosecond time stamps, uncertainty is inherent in them. Careful handling is required when working with them.

currentTimeMillis(), for example, is usually synchronized between machines reasonably well, and it can be used to measure network latencies, but nanoTime() is not useful between machines.

GET READY, GET SET

Getters and setters are simple methods that are much more expensive if they are not inlined, because the call is more expensive than the field access—a prime candidate for inlining.

To put some of the above theory into practice, we're going to look at a very simple (but extremely powerful) JIT compilation technique.

Method Inlining

One of the key optimizations the JIT compiler (but not javac) does is *method inlining*: copying the body of methods into the methods that call them and eliminating the call. This functionality can be important, because the cost of calling into a trivial method can be expensive compared to the work done in it.

The JIT compiler is able to progressively inline—that is, start by inlining simple methods and then move on to larger and larger blocks of code as other optimizations become possible.

The example shown in **Listing 1**, **Listing 1A**, and **Listing 1B** is a simple test harness that compares the performance of using a field directly or via a getter/setter method.

Getters and setters are simple methods that are much more expensive if they are not inlined, because the call is more expensive than the field access—a prime candidate for inlining.

If you run the test harness using java -cp . -XX:PrintCompilation Main, you can see the difference in performance (see **Listing 2**).

What does all this mean? The first column in **Listing 2** is the number of milliseconds since the program started. The second column is the method ID (for compiled methods) or the itera-

JVM Convergence

Oracle engineers are currently working to merge Java HotSpot VM and Oracle JRockit into a converged offering that leverages the best features of each. Oracle plans to contribute the results of the combined Java HotSpot and Oracle JRockit Java Virtual Machines (JVMs) into OpenJDK. Here are the key points:

- Oracle JRockit and HotSpot will be merged into a single JVM, incorporating the best features from both.
- The converged JVM will be based on HotSpot code, with features from Oracle JRockit included.
- The result will be contributed incrementally to OpenJDK.
- Some existing value-adds, such as those in Oracle JRockit Mission Control, will remain proprietary (and licensed commercially).
- Oracle will continue to distribute free JDK and JRE binaries, which include some closed source items.
- The JVM convergence will be a multiyear process.

For more details about the JVM merge, read "Oracle's JVM Strategy" by Henrik Ståhl, senior director of product management for the Java Platform Group at Oracle. To learn more about HotSpot, visit the OpenJDK HotSpot page. You can see a complete list of JDK Enhancement Proposals, including on the converged JVM, at the JEP Index. To follow development and reviews of the merged JVM, you can join the hotspot-dev@openjdk.java.net mailing list.

—Tori Wieldt

tion count (for the number of iterations performed so far).

Note: The String and UTF_8 classes are not used directly by the test, but compilation output still appears for them because they're used by the platform.

On the second line in **Listing 2**, you can see that both tests are very slow. This is because the first run of the code includes the time to load each class. The next line is much faster even though no code tested is compiled at this stage.

Also note the following:

- At 1,000 and 5,000 iterations, direct access to the fields is faster than via getter/setter methods because the getter and setter have not been inlined or even optimized. Even so, both are pretty fast.
- By 9,000 iterations, the getter is optimized (because it is called twice per loop), which gives a slight overall improvement to performance.
- By 10,000 iterations, the setter has been optimized. The extra time spent including the optimized code means the code briefly runs slower.
- Finally, both test classes are optimized:
  - DFACaller uses direct access to the fields, and GetSetCaller uses the getter and setter. This is the point at which the getter and setter are not just optimized; they are also inlined.
  - You can see that in the next iterations, the test times are still not optimal.
- After 13,000 iterations, the performance for each is as good as the final, much longer test. We've reached steady-state performance.

The important thing to note is that the steady-state performance for accessing fields directly or using getters and setters is basically the same because the methods have (finally) been inlined into GetSetCaller, meaning the callable code in viaGetSet is doing exactly the same work as the code in directCall (which accesses the fields directly).

The JIT compilation is performed in the background, and exactly when each optimization is available for execution varies from machine to machine and, somewhat, from run to run.

Conclusion

In this article, we've shown you the very tip of the JIT compilation iceberg. In particular, we haven't addressed some very important aspects of how to write good benchmarks and how to use statistics to ensure that the dynamic nature of the platform isn't fooling you.

The benchmark used here is very simple, and it isn't suitable for a real benchmark. In Part 2, we plan to show you how to handle a more realistic benchmark and also delve deeper into the code that the JIT compiler produces when it compiles your code. `</article>`

LEARN MORE

- Java HotSpot VM
- Just-in-time compilation

LISTING 1 LISTING 1A / LISTING 1B / LISTING 2

```
public class Main {

  private static double timeTestRun(String desc, int runs, Callable<Double> callable)
throws Exception {
    long start = System.nanoTime();
    callable.call();
    long time = System.nanoTime() - start;
    return (double) time / runs;
  }

  // Housekeeping method to provide nice uptime values for us
  private static long uptime() {
    return ManagementFactory.getRuntimeMXBean().getUptime() + 15;
  // fudge factor
  }

  public static void main(String... args) throws Exception {
    int iterations = 0;
    for (int i : new int[]{ 100, 1000, 5000, 9000, 10000, 11000, 13000, 20000,
100000} ) {
      final int runs = i - iterations;
      iterations += runs;

      // NOTE: We return double (sum of values) from our test cases to
      // prevent aggressive JIT compilation from eliminating the loop in
      // unrealistic ways
      Callable<Double> directCall = new DFACaller(runs);
      Callable<Double> viaGetSet = new GetSetCaller(runs);

      double time1 = timeTestRun("public fields", runs, directCall);
      double time2 = timeTestRun("getter/setter fields", runs, viaGetSet);

      System.out.printf("%7d %,7d\t\tfield access=%.1f ns, getter/setter=%.1f ns%n",
uptime(), iterations, time1, time2);
      // added to improve readability of the output
      Thread.sleep(100);
    }
  }
}
```

Download all listings in this issue as text

JAMES L. WEAVER

BIO

Part 1

Using Properties and Binding in JavaFX 2.0

Implement the power of binding in your JavaFX applications.

JavaFX 2.0, released in October 2011, is an API and runtime for creating Rich Internet Applications (RIAs). One of the advantages of JavaFX 2.0 is that the code can be written in the Java language using mature and familiar tools.

This two-part series focuses on JavaFX 2.0 properties and binding, which are often used to keep the state of the user interface (UI) in sync with the application's model.

JavaFX 2.0 comes with a set of interfaces, shown in **Figure 1**, whose purpose is to provide support for using and implementing properties, detecting when the values of properties have changed, and binding properties to other properties. These interfaces are located in four packages: javafx.beans, javafx.beans.binding, javafx.beans.property, and javafx.beans.value. This article contains an example of using the methods defined by many of these interfaces to use properties and binding.

Figure 1

PHOTOGRAPH BY STEVE GRUBMAN

GET TOGETHER

Properties and binding keep the UI and application model in sync.

Overview of the BindingExampleSolution Application

To help you learn how to use properties and binding, an example application named BindingExampleSolution will be employed. As shown in **Figure 2**, this application contains three slider controls and numeric values that you'll keep in sync with the sliders a little later.

The BindingExampleExercise project that you'll download in the next section contains starter code for the example application. In its current form, the application's runtime appearance is similar to **Figure 2**. During the course of this article, you'll modify the code to implement the binding behavior of the BindingExampleSolution application, which is shown in **Figure 3**.

As shown in **Figure 3**, when you move the sliders labeled InvalidationListener and ChangeListener, the values of the labels to their right reflect the value of the corresponding slider from 0.0 through 100.0.

Also, when you select the Bind checkbox and move the slider labeled bind/unbind, the value to its right reflects the value of the bind/unbind slider added to the product of the value of the top two sliders.

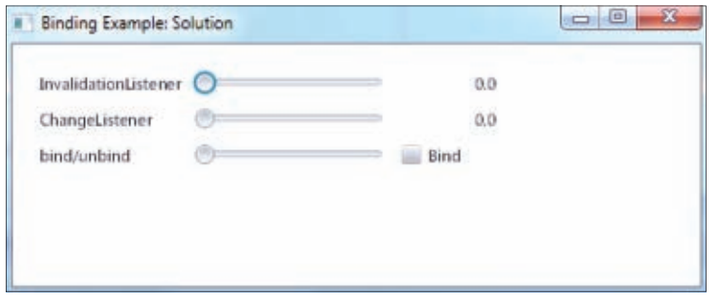


Figure 2

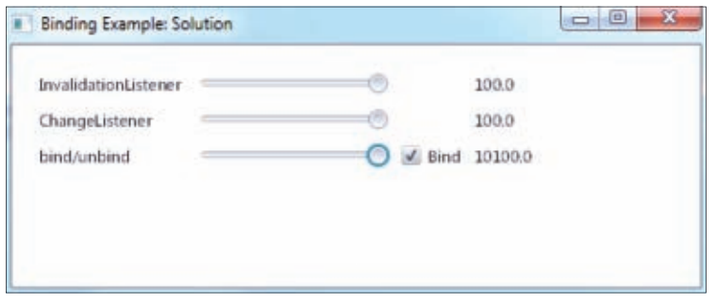


Figure 3

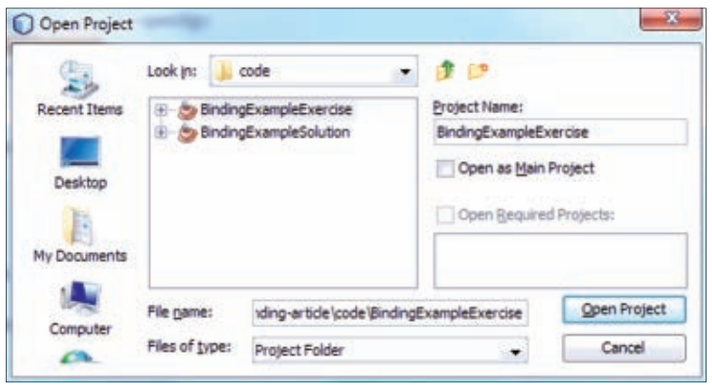


Figure 4

Figure 5

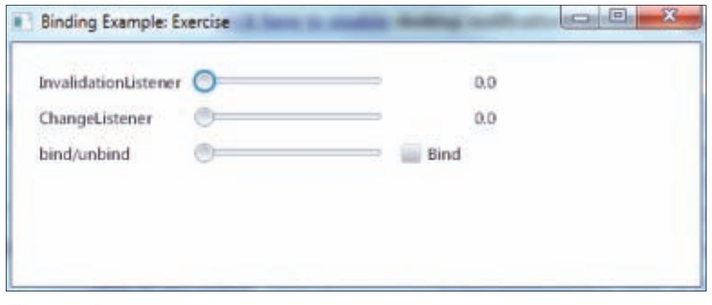


Figure 6

Obtaining and Running the BindingExampleExercise Project

1. Download the NetBeans project file, which includes the BindingExampleExercise project.
2. Expand the project into a directory of your choice.
3. Start NetBeans, and select **File -> Open Project**.
4. From the Open Project dialog box, navigate to your chosen directory and open the BindingExampleExercise project, as shown in **Figure 4**. If you receive a message stating that the jfxrt.jar file can't be found, click the **Resolve** button and navigate to the rt/lib folder subordinate to where you installed the JavaFX 2.0 SDK.

Note: You can obtain the NetBeans IDE from the NetBeans site.

5. To run the application, click the **Run Project** icon on the toolbar, or press the **F6** key. The Run Project icon looks like the Play button on a DVD player, as shown in **Figure 5**.

The BindingExampleExercise application should appear in a window, as shown in **Figure 6**.

LISTING 1

```
final DoubleProperty valueA = new SimpleDoubleProperty(0);

Slider sliderA = SliderBuilder.create()
 .max(100)
 .build();
sliderA.valueProperty().bindBidirectional(valueA);
```

Download all listings in this issue as text

Notice that moving the sliders has no effect. Your mission will be to add code that implements the behavior described previously. The next sections describe the steps you can follow to implement this behavior.

Step 1: Adding an InvalidationListener to Detect Changes to the Value of the Top Slider

To dynamically update the label text as the top slider is moved, we'll add an InvalidationListener to a property that tracks with the value of the slider.

Take a look at the code in BindingExampleMain.java in the BindingExampleExercise project, which shows the starter code for this example. We'll show code snippets from BindingExampleMain.java as you perform the steps in this exercise.

Instantiate a SimpleDoubleProperty that tracks with the value of the slider. The starter code in BindingExampleMain.java instantiates a SimpleDoubleProperty and bidirectionally binds the value property of the slider with it, as shown in **Listing 1**.

The SimpleDoubleProperty class is one of several classes (for example, SimpleBooleanProperty) in the javafx.beans.property package that can create properties that represent most of the Java datatypes.

The value property returned by the valueProperty() method of the Slider class is of type DoubleProperty. DoubleProperty is an abstract class that implements the Property interface shown in the Unified Modeling Language (UML) diagram in **Figure 1**, which contains a bindBidirectional() method.

We're using this bindBidirectional() method to bidirectionally bind the value property of the slider to the DoubleProperty referenced by the valueA variable. As a result, when the user moves the slider, the value of the DoubleProperty referenced by valueA will track with the value property of the slider. Conversely, if the value of the DoubleProperty referenced by the valueA variable is changed, the value property of the slider will change as well.

Add an InvalidationListener that updates the text property of the label. As shown in the UML diagram in **Figure 1**, the Property class indirectly extends the Observable interface. The code shown in **Listing 2** from BindingExampleMain.java leverages the addListener() method

defined in the Observable interface to add an InvalidationListener to the DoubleProperty referenced by valueA.

This invalidated() method is invoked whenever the valueA property is no longer valid, which means that the value *might* have changed. To update the label to the right of the slider, we'll put some code in the invalidated() method that casts the Observable object to a DoubleProperty, gets its value, converts it to a string, and sets the text of labelA to the string.

Go ahead and fill in the lines indicated by the "TO DO" comments, so the code in **Listing 2** turns into the code shown in **Listing 3**.

Note: This use of an InvalidationListener is for demonstration purposes, showing that the Observable argument passed into the invalidated() method can be read with the getValue() method. The act of reading the value, however, forces it to be reevaluated, because the implementation of SimpleDoubleProperty uses *lazy evaluation*. This result defeats the purpose of using an InvalidationListener, which is to be notified that a value is no longer valid without forcing it to be reevaluated. In contrast, the purpose of the ChangeListener (which you'll experience next) is to read the value as soon as it changes.

Now that you've implemented the invalidated() method, run the application to try the slider labeled InvalidationListener. After doing so, let's address the middle slider, which is labeled ChangeListener.

Jim Weaver introduces the concept of binding in JavaFX 2.0.

Step 2: Adding a ChangeListener to Detect Changes to the Value Property of the Middle Slider

To dynamically update the label text as the middle slider is moved, this time we'll add a ChangeListener to the property that tracks with the value of the middle slider.

As in Step 1, the starter code for the middle slider in BindingExampleMain.java instantiates a SimpleDoubleProperty and bidirectionally binds the value property of the slider with it, as shown in **Listing 4**.

Add a ChangeListener that updates the text property of the label. As shown in the UML diagram in **Figure 1**, the Property class indirectly extends the ObservableValue interface. The code shown in **Listing 5** from BindingExampleMain.java leverages the addListener() method defined in the ObservableValue interface to add a ChangeListener to the DoubleProperty referenced by valueB.

This changed() method is invoked whenever the valueB property has changed. To update the label to the right of the slider, we'll put some code in the changed() method that converts the newValue to a string and sets the value of the text property of labelB to the string.

Go ahead and fill in the lines indicated by the "TO DO" comments, so the code in **Listing 5** turns into the code shown in **Listing 6**.

You might recall that in **Listing 3**, we used the setText() method of labelA to set its text property. To demonstrate an alternative, here we're using the textProperty() method of labelB to get the underlying StringProperty and we're using its setValue() method to set the text.

Now that you've implemented the changed() method, run the application again to try the slider labeled ChangeListener. After doing so, let's address the bottom slider, which is labeled bind/unbind.

Step 3: Using bind(), unbind(), and Bind Expressions to Manage the Text Property of the Bottom Label

To dynamically update the label text as the bottom slider is moved, this time we'll bind the text property of the label to the properties that track with the values of the sliders.

As in Step 2, the starter code for the bottom slider in BindingExampleMain.java instantiates a SimpleDoubleProperty and bidirectionally binds the value property of the slider with it, as shown in **Listing 7**.

Add a ChangeListener to the checkbox. As shown in **Figure 3**, there is a checkbox labeled Bind that, when selected, causes

LISTING 2 LISTING 3 LISTING 4 LISTING 5 LISTING 6 LISTING 7

```
valueA.addListener(new InvalidationListener() {
  public void invalidated(Observable observable) {
    // observable is a DoubleProperty
    // TO DO: Cast the Observable object to a DoubleProperty; get its value
    //      and convert it to a String, and set the text of
    //      labelA to the String.

  }
});
```

Download all listings in this issue as text

the text property of the label to be bound to the value of the bottom slider added to the product of the value of the other two sliders.

Detecting when the status of the checkbox changes is handled by the code in **Listing 8** from BindingExampleMain.java.

Recall that we used a ChangeListener in Step 2, with the main difference being that newValue here is a Boolean rather than a Double because the ObservableValue in this case is a SimpleBooleanProperty.

Create bind expressions. The bind (actually, the bidirectional bind) from **Listing 7** binds one property to another property of the same type, with both properties holding the same value. It is often desirable to bind an expression that contains one or more properties to another property.

SERIOUS SUPPORT

JavaFX 2.0 comes with **a set of interfaces whose purpose is to provide** support for implementing properties, detecting when the values of properties have changed, and binding properties to other properties.

To demonstrate this technique, we'll put some code that uses a *bind expression* into the changed() method from **Listing 8**.

Go ahead and fill in the lines indicated by the "TO DO" comments, so the code in **Listing 8** turns into the code shown in **Listing 9**, and then run the code to verify that the bottom slider and the Bind checkbox exhibit the expected behavior.

LISTING 8 LISTING 9

```
checkBoxC.selectedProperty().addListener(new ChangeListener<Boolean>() {
 public void changed(ObservableValue ov,
            Boolean oldValue, Boolean newValue) {
  if (newValue.booleanValue()) {
   // TO DO: Using the bind() method and the Fluent API, bind the text
   // property of labelC to the product of valueA and valueB plus valueC
  }
  else {
   labelC.textProperty().unbind();
  }
 }
});
```

Download all listings in this issue as text

As shown in **Listing 9**, a bind expression consists of methods chained together that comprise the expression. These methods are inherited from classes that exist in the javafx.beans.binding package and, in this case, they are inherited by the DoubleProperty from the DoubleExpression and NumberExpressionBase classes.

Bind expressions are sometimes referred to as using the *fluent interface API*, which is a programming style in which a method chain reads like a fluent sentence.

Also shown in **Listing 9** is an unbind() method invocation, which breaks the bind that was previously established with the bind expression.

Conclusion

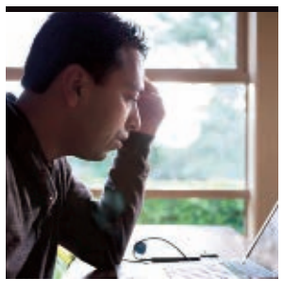
JavaFX 2.0 comes with numerous classes and interfaces that provide a powerful properties and bindings framework. This framework fires two kinds of events—invalidation events and change events—which are handled by the InvalidationListener and ChangeListener interfaces, respectively.

Properties can be bound to each other, either unidirectionally or bidirectionally. You can also leverage bind expressions to bind the value of one property to an expression that contains one or more properties.

In Part 2, we'll look at optimizing JavaFX 2.0 properties and bindings by implementing lazy initializations and custom bindings. `</article>`

LEARN MORE

- The properties and binding tutorial at Oracle's JavaFX site
- Michael Heinrichs' blog, which includes entries on JavaFX properties and bindings



VIKRAM GOYAL

BIO

How to Tag Your Pictures with Location Information

Learn how to insert GPS data into images taken with a mobile device.

Geotagging is a process by which you can add global positioning system (GPS) geographical metadata to images taken with a mobile device. Adding this metadata helps you to identify, at a later date, where a particular image was taken. Geotagging is not enabled by default in GPS-enabled devices, so you need to write your own special application to add the metadata.

In this article, I explain the process by which you can insert GPS metadata into an image, and I develop a rudimentary application you can use to insert GPS metadata into images.

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

Process for Geotagging

In a nutshell, by geotagging an image, you insert special GPS metadata in the file structure. Only some image types can be geotagged. Therefore, the process by which you obtain an image that is to be geotagged needs to be able to supply the image in the correct format. In this article, we will use the JPEG image type.

For the scope of this article, geotagging involves putting the latitude and longitude information within the image. However, with geotagging, you also can insert advanced data, such as altitude, azimuth, and even place names.

To geotag an image, you need to do the following:

1. Establish that location data is available (that is, determine that the mobile device's current latitude and longitude can be retrieved and used within the application).
2. Capture an image that will be geotagged.
3. Insert the GPS metadata into the image data.
4. Save the image, with the metadata intact, to the file system.

The first two steps are simple and can be done together by the same application. See "Using the Location API for Favorite Spots" and "Taking Pictures with MMAPI" for more information.

Steps 3 and 4 require a detailed examination of the process. But first, let's establish the location data.

GPS METADATA

Geotagging is a process by which you can add global positioning system (GPS) **geographical metadata to images** taken with a mobile device.

Establishing the Location Data

If you have not done so already, it might be useful to download the source code while you are reading the rest of this article.

To establish the location data, we will create a very simple Criteria option for the location provider, as shown in **Listing 1**.

We will then create the LocationProvider, as shown in **Listing 2**.

If the correct LocationProvider is found, we will set the current MIDlet as the listener, as shown in **Listing 3**. The code in **Listing 3** simply updates a class variable with the current location provided by the LocationProvider.

We have the current location data, so next, we need to establish a separate Canvas class for displaying the viewfinder and taking photos.

Creating the Viewfinder and Taking Photos

The viewfinder is created using MMAPI classes—specifically the VideoControl class. We migrate this class into its own separate Canvas class, called CameraCanvas. The constructor for this creates, initializes, and displays the VideoControl, as shown in **Listing 4**. Once the viewfinder is displayed, you can press the Fire key to take a snapshot, as shown in **Listing 5**.

This is where we intervene to modify the bytecode that is gen-

PHOTOGRAPH BY JONATHAN WOOD/ GETTY IMAGES

erated to add the location data. To do so, we first need to understand the file structure, because we will be modifying it at the byte level.

NOT JUST IMAGES

EXIF was originally created to tag any media with metadata information.

The EXIF Format

EXIF stands for *exchangeable image file format*. In spite of its name, it was originally created to tag any media with metadata information. It is based on the JPEG file interchange format (JFIF) and, more importantly, you can insert EXIF format data into existing JPEG images to tag them with the correct metadata.

You can think of the EXIF format as a special container within a JPEG file for holding metadata for that image. This container is also capable of holding thumbnail information so that the original JPEG image becomes more useful for display purposes.

You can easily identify a JPEG file that holds an EXIF metadata container by opening such a file within a Hex viewer and looking for the text *Exif*. The data that follows this text is what defines the EXIF structure.

However, even before you get to the *Exif* text, you will see items that the JPEG standard defines as *application markers*. These markers start with the special bytes `0xFFE0` through `0xFFEF`. While the application markers are not necessary for rendering the image data, they provide processing information. EXIF uses the special `0xFFE1` marker to indicate that the EXIF application marker is starting.

Following the `0xFFE1` marker, the length of the marker is specified using 4 bytes (including the marker). Finally, there is the special string *Exif*, which marks the start of the EXIF structure.

So far, we have the following structure for an EXIF encoded metadata image file, where `LLLL` is the total length of the EXIF application marker, `0x45` through `0x66` represent the string *Exif*, and the last two `0x00` bytes mark the end of this section:

```
0xFFE1 LLLL 0x45 0x78 0x69
0x66 0x00 0x00
```

DESCRIPTION OF BYTES	VALUE OF BYTES			
4 BYTES THAT TELL US HOW MANY DIRECTORY ENTRIES ARE IN THIS IFD:	0X0003			
THE FIRST DIRECTORY ENTRY:	TTTT	FFFF	NNNNNNNN	DDDDDDDD
THE SECOND DIRECTORY ENTRY:	TTTT	FFFF	NNNNNNNN	DDDDDDDD
THE THIRD AND FINAL DIRECTORY ENTRY:	TTTT	FFFF	NNNNNNNN	DDDDDDDD
8 BYTES THAT SHOW THE OFFSET TO THE NEXT IFD (OR 0X0000, IF THIS IS THE LAST IFD):	0X00000000			

Table 1

LISTING 1 | LISTING 2 | LISTING 3 | LISTING 4 | LISTING 5

```
Criteria criteria = new Criteria();
criteria.setHorizontalAccuracy(Criteria.NO_REQUIREMENT);
criteria.setVerticalAccuracy(Criteria.NO_REQUIREMENT);
criteria.setPreferredResponseTime(Criteria.NO_REQUIREMENT);
criteria.setCostAllowed(false);
```

Download all listings in this issue as text

Following this, there is a mandatory TIFF header structure:

```
0x49 0x49 0x2A 0x00
```

This leads to the next 4 bytes, which tell us where in this file structure we will find a directory of information for the image data. These 4 bytes are like an offset to the actual data. This data, which has a special structure, is called an image file directory (IFD).

Understanding the IFD

The last 4 bytes of the TIFF header are a pointer to the actual location of the first IFD. At the end of the first IFD, the last 4 bytes then point to the location of the next IFD, and so on until there are no more IFDs. The last IFD has `0x0000` as the last 4 bytes to signify that there are no more IFDs.

The IFD is like a compact directory of information. Each EXIF container can hold multiple directories that form a chain, with the first one pointing to the next through its last 4 bytes.

We are mainly interested in the GPS IFD because we want to insert our GPS metadata there, but all IFDs have the same structure, as shown in **Table 1**.

In **Table 1**, the first 4 bytes state that there are three directory entries within this IFD. Next are the three directory

entries, each with a structure that provides the data the directories hold. Finally, the 8 bytes at the end link to the next IFD, if there is one, or they contain all zeros, if there are no more IFDs. Typically, this would mark the end of the EXIF container as well.

Each directory entry follows the same structure. The first 4 bytes (TTTT) represent a tag number that is relevant to the type of IFD. Because we are concentrating on GPS IFDs, the first tag would be 0x0000, which represents the GPS version number. Thus, what we are saying is that this directory entry contains the GPS version number.

The next 4 bytes (FFFF) tell us the format of this data. There are 12 different types of data formats, from ASCII, to byte, to double float. Each type has its own number, for example, ASCII is represented by the number 2 and unsigned rational is represented by the number 5. Thus, if this were the directory structure for the GPS version number, these 4 bytes would contain the value 0x02 0x00.

The next 8 bytes are NNNNNNNN, and they tell us how many components are available for this value. Each data value might have multiple components that together make up the data. Because the GPS version number contains only a single component, we would just use the value 0x01 0x00 0x00 0x00.

Finally, the last 8 bytes (DDDDDDDD) are where the data is stored or they are the offset to where the data is stored. For the GPS version number, this is typically 0x32 0x32 0x00 0x00 (representing GPS version 2.2.0.0).

With this understanding of the EXIF file structure behind us, let's move on to the actual code.

Modifying the Image Data

In the prior sections, we gathered the location data and took a snapshot. We will gather this data in a byte array and proceed to modify it, as shown in **Listing 6**.

I haven't shown it here, but we are doing this process (including the taking of the snapshot) in a separate thread so we don't block the main application. (See the complete source code.)

Now we will go through this array one byte at a time and modify it accordingly, as shown in **Listing 7**.

The EXIF format uses the marker 0xE1. The image that we process using the Sun Java Wireless Toolkit has the marker 0xE0 to represent a pure JPEG image. We replace that with the E1 tag and then write the total size (which we calculate manually).

Then we write the familiar *Exif* string to identify that we are now writing an EXIF section. This is followed by the mandatory TIFF header. See **Listing 8**.

We then have the first IFD, the GPS IFD, as shown in **Listing 9**. Note that the GPS IFD doesn't contain the actual data, but an offset to the location of the actual GPS IFD structure, which starts with an indication of how many entries it contains (three: version, latitude, and longitude) followed by the first entry (the GPS version), as shown in **Listing 10**.

The next entry is the GPS latitude. See **Listing 11**. Notice that it doesn't actually contain the data. It links to the data via the offset.

I will skip the GPS longitude entries to show you the offset data (see **Listing 12**).

I used a special method to convert the GPS latitude to degrees and minutes from a decimal representation. The format expects the degrees and minutes that make up the GPS informa-

WRITE AN APP

Geotagging is **not enabled by default in GPS-enabled devices**, so you need to write your own special application to add the metadata.

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

```
imageArray = videoControl.getSnapshot("encoding=jpeg");
// read it
ByteArrayInputStream bis = new ByteArrayInputStream(imageArray);
// for writing the modified array
ByteArrayOutputStream bos = new ByteArrayOutputStream();
```

Download all listings in this issue as text

LISTING 11 LISTING 12

```
// second entry is GPS Latitude - 2 bytes
bos.write(0x02);
bos.write(0x00);

// data format (rational) - 2 bytes
bos.write(0x05);
bos.write(0x00);

// number of components (2 - degrees and minutes) - 4 bytes
bos.write(0x02);
bos.write(0x00);
bos.write(0x00);
bos.write(0x00);

// offset to data value (64) - 4 bytes
bos.write(0x40);
bos.write(0x00);
bos.write(0x00);
bos.write(0x00);
```

Download all listings in this issue as text

tion to have their own entry (that is why we specified earlier that the number of components for the GPS latitude is 2). We could become even more fine-grained and show seconds as well. In that case, we would need to mark the number of components as 3.

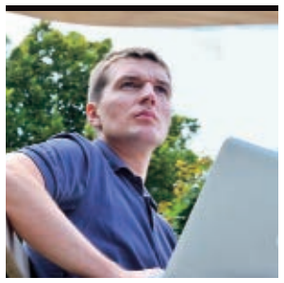
The GPS longitude uses the same format (omitted here for space purposes).

Conclusion

This article combined three separate technologies to modify an image to hold GPS metadata information. We touched only briefly on the Location API and taking snapshots. We then discussed modifying the image data to hold EXIF metadata information, covered the basics of EXIF, and worked on the code for an actual application. `</article>`

LEARN MORE

- "Getting Started with Touchscreen UIs in Java Platform, Micro Edition"
- "Working with the XML Parser API — JSR 172"
- "Working with the Mobile Sensor API"



ADAM BIEN

BIO

Convention over Configuration in Java EE 6

Learn to specify the unconventional through Convention over Configuration.

Convention over Configuration in Java EE 6 requires you to specify only the unconventional. Obvious choices do not have to be configured. However, some of the environmental or deployment-dependent choices simply cannot be derived from the convention and have to be configured. This article focuses on project-specific or product-specific configuration of Java EE 6 applications that require explicit configuration.

Staging

Information about the current stage (such as test, integration, or production) is particularly interesting for applications with high amounts of legacy code. Legacy connectors often are not fully Java EE–compliant and cannot use Java EE resource management. Instead of relying on the application server's configuration capabilities, you have to mock out external resources in the preproduction stages.

A Stage can be represented as an enum and injected to all interested resources:

```
public enum Stage {
   Development, UnitTest,
SystemTest, Production
}
```

The current stage can be explicitly configured as a system property, configuration file, database, or any resource you prefer. JavaServer Faces 2 (JSF 2) supports a configurable staging environment with the ProjectStage class, so we can expose the existing information with a managed bean (see **Listing 1**).

The ProjectStage class originates from the JSF 2 API and belongs to the UI layer. To decouple the business logic implementation from the UI, the FacesContextExposer managed bean (**Listing 1**) converts the ProjectStage into a neutral Stage enum. The method FacesContextExposer#stage() "produces" the Stage and makes it available for injection. The staging value can be configured in the JNDI tree with the key java:comp/env/jsf/ProjectStage or with the context-param tag in the web.xml deployment descriptor (see **Listing 2**). The JNDI key takes precedence—it overwrites the staging value configured in web.xml.

NOT SO OBVIOUS

Convention over Configuration in Java EE 6 **requires you to specify only the unconventional.** Obvious choices do not have to be configured.

With the built-in annotation @Produces exposed, the Stage enum is directly injected as a Stage projectStage parameter of the ConnectorManager#connect method (see **Listing 3**).

Inside the connect method, the Stage enum is used to decide which EISConnector implementation to create. Only in the case of Stage.Production is the LegacyConnector created. For all other stages, EISMock is instantiated and returned. To distinguish a staged injection from the "usual" injection, the connect method is annotated with the @StageDependent qualifier (see **Listing 4**).

A Qualifier "marker" has no attributes and is used only to extend the type of the injected object (see **Listing 5**).

The user of EISConnector only has to use the @StageDependent qualifier. The code can be read fluently: "Inject the stage-dependent EISConnector."

The injected EISConnector inherits the scope of the user. In our case, the user is the Index managed bean denoted with the @Model annotation. The @Model annotation is an @Stereotype (a macro), which is "expanded" to the

PHOTOGRAPH BY THOMAS EINBERGER/ GETTY IMAGES

@RequestScoped and @Named annotations. On each request, a new instance of the EISConnector is injected to the Index managed bean. A stateless session bean would cause only a single injection per pooled instance.

Configuration with Convention

In addition to stages, any arbitrary primitive (String, int, long, float) and other objects can be configured dynamically. Let's assume that an imaginary customer expects us to build a highly configurable Greeter component (see **Listing 6**).

The Greeter stateless session bean returns a configurable message that is concatenated a configurable number of times. We simply use the @Inject annotation to inject primitive members, as shown in **Listing 6**.

The injection of the Greeter#message and Greeter#repetition attributes immediately causes the deployment error shown in **Listing 7**.

The "unsatisfied dependencies" error indicates the lack of adequate instances for the injection. We have to make the primitive available for injection first by using the @Produces annotation. The easiest possible solution for the "unsatisfied dependencies" problem is direct exposure of attributes with matching types in a dedicated class:

```
public class Configurator{
  private String msg =
"configurable";
  private int repetition = 2;
}
```

Although the introduction of the Configurator class fixes the problem for our simplistic demands and centralizes all configuration data in one place, this approach is not usable in real projects. You could configure only a single String that way. The Contexts and Dependency Injection (CDI) runtime matches the attribute types, not the names. Regardless of the attribute's name, all Strings would get the same value produced in the Configurator.

Any attempt to produce an additional String would result in exactly the opposite problem: an "Ambiguous dependencies" error. A custom @Qualifier would fix the ambiguity problem, but that approach would require you to write a custom @Qualifier annotation for each injected configurable type.

The built-in @Qualifier javax.inject.Named uses a String for disambiguation of different injection points, but that introduces another problem. A String-based dependency solution becomes hard to refactor and is, therefore, hard to maintain. The JSR 299 specification states the following in Chapter 3.11:

"The use of @Named as an injection point qualifier is not recommended, except in the case of integration with legacy code that uses string-based names to identify beans."

Instead of introducing explicit qualifiers, we could rely on convention and derive the configuration key from the injection point's name. The javax.enterprise.inject.spi.InjectionPoint instance represents the injection point and is available for a producer method. With the necessary type name of the attribute with its declaring class extracted from the InjectionPoint, we could dynamically configure the whole system (see **Listing 8**).

WATCH OUT FOR THIS

A dependency between the configuration and the fully qualified class names can cause inconsistencies.

Now the name of the injection point is used to fetch the corresponding value from a given configuration store. A simple attribute's name might not be unique enough in a larger project, so you could prepend the fully qualified class name to the attribute's name to make it globally unique. The fully qualified class name can be obtained from the InjectionPoint as well (see **Listing 9**).

The conventional use of the type information for attribute configuration requires the fully qualified names of the attributes to be maintained in the configuration store. A dependency between the configuration and the fully qualified class names can cause inconsistencies if the injection point changes its name or the enclosing class moves to another package. For this reason, an additional annotation is introduced to set the name of the configuration key explicitly (see **Listing 10**).

Configurable is a Plain Old Java Annotation with a single attribute, String value(), and is fully independent of the Java EE API. Because Configurable is not denoted with any CDI meta-annotation, such as @Qualifier or @Stereotype, the application server ignores it.

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

```
@Model
public class FacesContextExposer {
  @Produces
  public Stage stage(){
    FacesContext ctx = FacesContext.getCurrentInstance();
    return Stage.valueOf(ctx.getApplication().getProjectStage().name());
  }
}
```

Download all listings in this issue as text

We are using the Configurable annotation to override the configuration key. If a field is denoted with Configurable, we are using the value() as a lookup key instead of using the fully qualified field name (see **Listing 11**).

The configuration consumer only has to provide the desired lookup key by denoting the field with the @Configurable annotation and providing the value():

```
@Stateless
public class Greeter {
  @Inject @Configurable
("greetings")
  private String message;
//..
}
```

Stage-Dependent Configuration

The injection of primitives and the staging mechanism can be combined easily. The already introduced @StageDependent qualifier can be used to produce stage-dependent configuration entries. Also, the exposed Stage is injected to the getString method and used to build a stage-dependent key, as shown in **Listing 12**.

TAKE CONTROL

In addition to overriding the defaults with plug-ins, **you can even manage the configuration at runtime.**

With the prepended stage name, the field name or the value of the @Configurable annotation is used to fetch the configuration value. Now we are not only able to inject different implementations of an interface, but we can also provide stage-dependent configuration via injection of primitive types:

```
@Stateless
public class Greeter {
  @Inject @StageDependent
  private String stagedMessage;
}
```

The consumer only has to mark the injected field as @StageDependent to get environment-dependent values.

Conventional Configuration Store

The centralization of code dealing with configuration is the main benefit of the approach discussed here. All configuration requests are served from a single place: the Configurator class. We could apply the "Convention over Configuration" spirit to the Configurator also and provide the default values for all obvious configuration entries out of the box (see **Listing 13**).

In the fetchConfiguration method of the @Singleton Configurator, a java.util.HashMap is populated with default values. In our sample, a single Map is used for all stages, but it could be split easily into dedicated Map instances for each stage. Centralizing suitable defaults in a central Map simplifies maintenance. To change the values, however, you will have to recompile the Configurator class, which might not be desired for volatile entries.

Plug-ins (or the Sky Is the Limit)

By encapsulating the configuration population with the ConfigurationProvider interface, we introduce a flexible extension mechanism:

```
public interface
ConfigurationProvider {
  public Map<String,String>
getConfiguration();
}
```

The ConfigurationProvider interface is injected to the Configurator wrapped with the javax.enterprise.inject.Instance<ConfigurationProvider> interface. With javax.enterprise.inject.Instance<ConfigurationProvider>, the injection of the ConfigurationProvider interface is lazy and will work even if no valid implementation of the interface is deployed (see **Listing 14**).

Interestingly, the javax.enterprise.inject.Instance inherits from the java.lang.Iterable interface, which allows iteration over all implementations in an ordinary loop. We loop over all found implementations (see the Configurator#mergeWithCustomConfiguration), fetch the external Map<String,String> instance, and merge it with the default values hardcoded in the @PostConstruct method. It is the simplest possible implementation of a plug-in mechanism in Java EE: you can easily extend existing functionality with external implementation on demand.

Because we already relied on JSF to provide the staging information, we could also use JSF as a configuration provider. We only have to expose the Servlet's context-params as a Map<String,String> in the FacesContextExposer (see **Listing 15**).

In the next step, we implement the ConfigurationProvider interface and inject it with the FacesContextExposer provided by the Map instance (see **Listing 16**).

Now the hardcoded default values can be overridden easily in web.xml by context-param entries (see **Listing 17**).

You are not limited to web.xml. Other implementations of the ConfigurationProvider interface could fetch the implementation from XML, property files, or databases with the Java

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

```
org.jboss.weld.exceptions.DeploymentException: org.jboss.weld.exceptions.
DeploymentException: WELD-001408 Unsatisfied dependencies for type [String]
with qualifiers [@Default] at injection point [[field] @Inject private com.abien.
configuration.business.primitives.consumer.Messenger.message] [...]
```

Download all listings in this issue as text

Persistence API. In any case, you can plug in your resources with a few lines of code. And best of all, if the hardcoded values do meet your expectations, you do not have to implement anything. The Configurator also works without any ConfigurationProvider implementations deployed.

More Control with JAX-RS and JMX

In addition to overriding the defaults with plug-ins, you can even manage the configuration at runtime. The Configurator is implemented as an @Singleton bean and can be exposed easily via representational Java API for RESTful Web Services (JAX-RS), as shown in **Listing 18**.

With the @Path("configuration") annotation, the Configurator becomes available under the following URI:

> *[http://localhost:8080/configuring-javaee]*/resources/configuration

With a plain HTTP GET request, you will get the whole configuration (method Configurator#getConfiguration), and with /configuration/{id} (method Configurator#getEntry), you will get a single entry.

An HTTP PUT /configuration/{id} request creates a new entry or overrides an existing one, and a DELETE request to URI /configuration/{id} (Configurator#deleteEntry) removes the existing entry.

To expose the configuration to Java Management Extensions (JMX), we only have to introduce an interface with

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

```
@Singleton
public class Configurator {
  @Inject
  private Instance<ConfigurationProvider> configurationProvider;
  private Map<String, String> configuration;
  @PostConstruct
  public void fetchConfiguration() {
    this.configuration = new HashMap<String, String>() {{
      put("message", "-configurable-");
      put("greetings", "-highly configurable-");
      put("Development.stagedMessage", "Development: -highly configurable-");
      put("repetition", "2");
      put("debug", "false");
    }};
  }
}
```

Download all listings in this issue as text

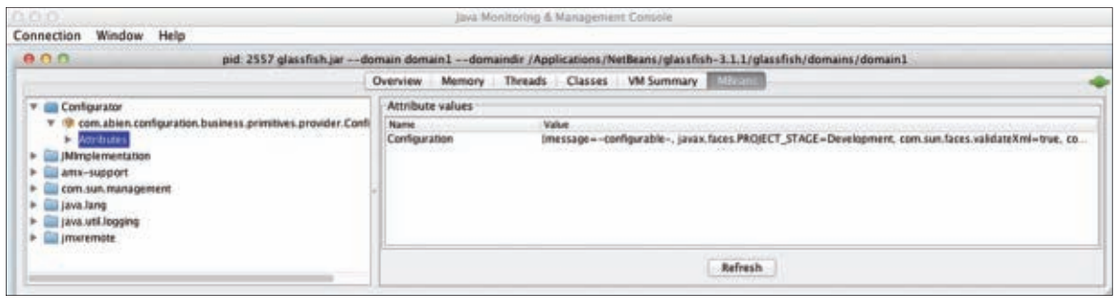


Figure 1

the MXBean ending and register the Singleton instance at the JMX runtime:

```
public interface
ConfiguratorMXBean {
  String getConfiguration();
}
```

The Configurator first has to be registered at the JMX runtime. The @PostConstruct method Configurator#fetchConfiguration invokes the Configurator#registerInJMX, which performs the JMX registration (see **Listing 19**).

Equally important is the unregistration of the Configurator at shutdown time, destroy time, or undeploy time. The method Configurator#unregisterFromJMX unregisters the Configurator from the MBeanServer. Without this important task, you can't redeploy your application. After the deployment, the Configurator with its configuration becomes visible in the Java Monitoring and Management Console (JConsole), as shown in **Figure 1**.

Keeping Things Consistent

The coupling between the attributes and the configuration store is loose. You can accidentally change a name of the attribute or the name of the key in the configuration, which could break the system at runtime.

Our centralized approach allows us to track all unsatisfied requests for configuration. Because all requests are served by the Configurator#getString method, we only have to keep track of all fields that couldn't be configured (see **Listing 20**).

The names of the fields without a corresponding configuration entry are stored in the HashSet<String> unconfiguredFields, and they are exposed with a public getter. Alternatively, we could just throw an exception to notify the user about a possible inconsistency.

Think About the Stakeholders

With Java EE 6, it is trivial to configure all components of an application. It is even possible to change the configuration on every request/transaction and change it via REST, JMX, SOAP, IIOP, or any other available protocol.

Before you eagerly make everything in your system configurable, first think about the stakeholders and the real requirements. Usually, no one wants (or actually dares) to change configuration entries in the running system; if you change the configuration entries, you will have to retest and redeploy your application anyway.

In this case, it is easier to hardcode the entries and build the whole system in a reliable way than to maintain the configuration externally and implement change processes and administration UIs. </article>

LISTING 19 LISTING 20

```
@Path("configuration")
@Produces(TEXT_PLAIN)
@LocalBean
@Singleton
public class Configurator implements ConfiguratorMXBean {

  private ObjectName objectName;
  private MBeanServer platformMBeanServer;

  @PostConstruct
   public void fetchConfiguration() {
   //...
    mergeWithCustomConfiguration();
    registerInJMX();
  }

  void registerInJMX(){
   try {
      objectName = new ObjectName("Configurator:type=" + this.getClass().getName());
      platformMBeanServer = ManagementFactory.getPlatformMBeanServer();
      platformMBeanServer.registerMBean(this, objectName);
    } catch (Exception e) {
      throw new IllegalStateException("..." + e);
    }
  }

  @PreDestroy
  public void unregisterFromJMX() {
    try {
      platformMBeanServer.unregisterMBean(this.objectName);
    } catch (Exception e) {
      throw new IllegalStateException("...:" + e);
    }
   //...other methods omitted
  }
```

Download all listings in this issue as text

//fix this /

In the last issue, Jonathan Giles showed us Java code that demonstrates a thread retrieving from a remote datasource, and updates a JavaFX ProgressBar control. He asked if this code was guaranteed to work in all environments.

The correct answer is #3: No. The ProgressBar must always be updated from the JavaFX application thread. Whenever you use separate threads in JavaFX, you must always be sure to update the scene only via the JavaFX application thread. This is achieved by calling Platform.runLater(..) from within your threading code. If this is not done, the user interface may not appear as expected in all circumstances (and on all machines). Future JavaFX releases may even make this a runtime exception, so do the right thing now!

This issue's challenge comes from Angela Caicedo, a Java evangelist at Oracle.

1 THE PROBLEM

One of the coolest features of JavaFX is binding capabilities. When JavaFX properties participate in bindings, changes made to one object are automatically reflected in another object. This is particularly useful when creating graphical user interfaces because you can automatically keep the display synchronized with the underlying data.

2 THE CODE

Consider the following code fragment for a JavaFX application:

```
...
IntegerProperty myInt = new SimpleIntegerProperty(5);
IntegerProperty myBoundInt = new SimpleIntegerProperty(0);
myBoundInt.bind(myInt);
myInt.set(8);
System.out.println("My bound integer's value is: "+ myBoundInt);
myBoundInt.set(5);
System.out.println("My bound integer's value is: "+ myBoundInt);
```

3 WHAT'S THE FIX?

What will be the output when you try to run this code?

1)
```
My bound integer's value is: 8
My bound integer's value is: 5
```

2)
```
My bound integer's value is: 5
My bound integer's value is: 5
```

3)
```
My bound integer's value is: 8
My bound integer's value is: 8
```

4)
```
My bound integer's value is: 8
Exception***
```

Hint: Read the JavaFX 2.0 documentation "Using JavaFX Properties and Binding."

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

Senior Design Director
Suemi Lam

Design Director
Richard Merchán

Contributing Designers
Jaime Ferrand

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