Java™ magazine

By and for the Java community

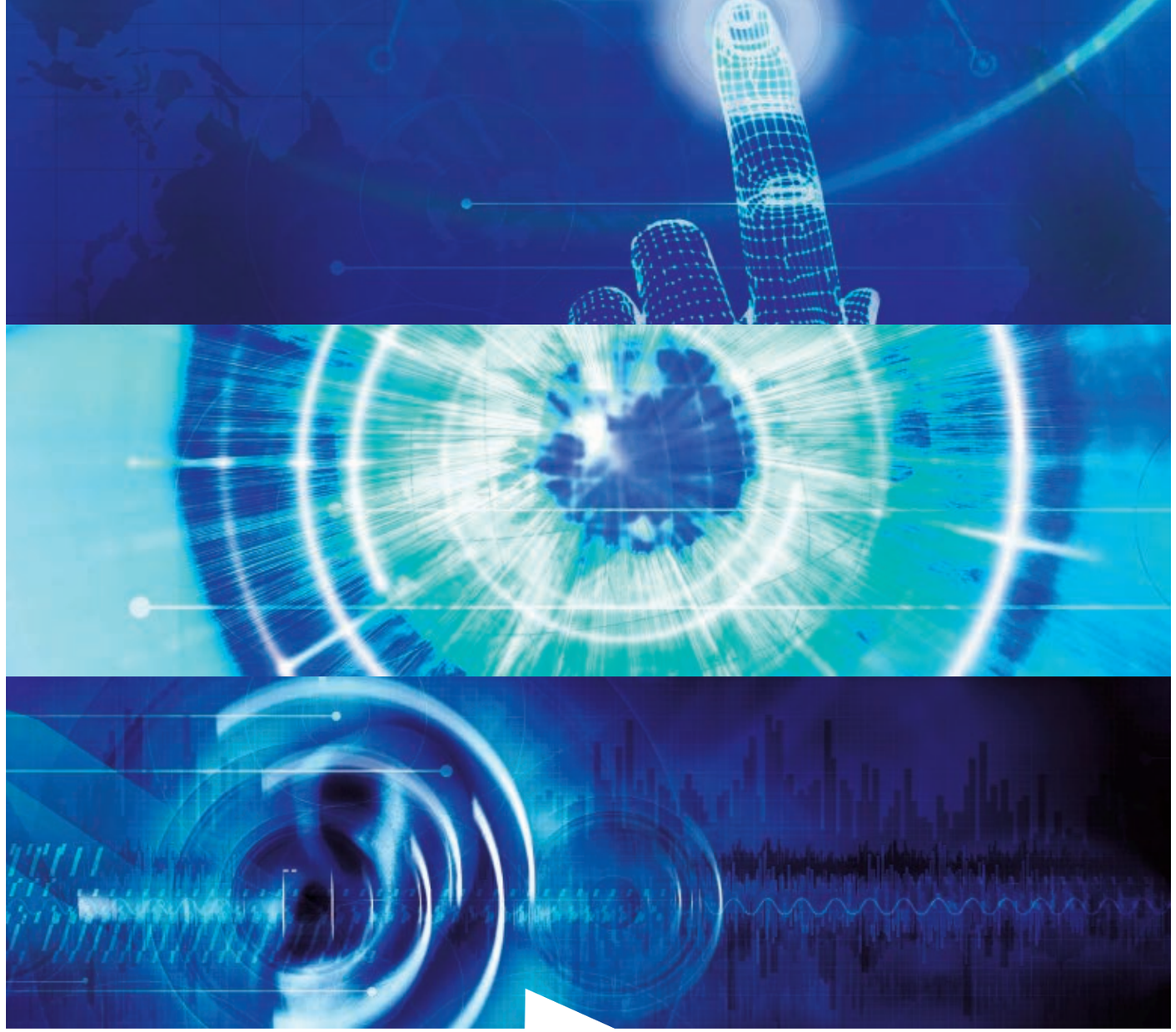
SHOCK THE SENSES

ORACLE.COM/JAVAMAGAZINE

ORACLE

//table of contents /

COVER ART BY PAULINA MCFARLAND



The previous (and Premiere) issue of *Java Magazine* published synchronously with the on-time general availability of Java SE 7. Our timing for this issue is almost as good, as production wrapped up just a couple of weeks after the conclusion of JavaOne 2011, in San Francisco, California.

If you were unable to attend, you've probably heard that JavaOne was exciting for a number of reasons, many of which you will read about in this issue. But if I had to choose the most notable theme, it would be the re-energized, reinvigorated, and resurgent JavaFX platform.

JavaFX 2.0, now generally available for Windows (and in Developer Preview mode for Mac OS X), allows you to write apps in plain old Java using standard development tools—unlike 1.0, which required knowledge of a custom scripting language. Furthermore, at JavaOne, Oracle announced its intention to fully open source the platform via the OpenJDK Project, and pursue its standardization via the Java Community Process. (Read all about these developments, and more, in our JavaFX coverage in this issue.) Looks like Java developers working client-side will finally get their next-generation successor to Swing.

There's more to the story though. At JavaOne, we also got the updated roadmaps for Java SE 8, Java EE 7, and Java ME. We got word of the JDK 7 for Mac OS X Developer Preview. And we were reminded that Java developers "code hard in their cubicles." Yes, it was quite a show.

So what's next? Well, we need to get you ready for Java EE 7 (scheduled for release in 2012), which will provide a genuinely standard runtime for cloud-based apps. Look for that in the January/February 2012 issue.

Justin Kestelyn, Editor in Chief BIO

//send us your feedback /

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

PHOTOGRAPH BY BOB ADLER

Hasan Rizvi

JavaOne 2011 Brings Focus, Momentum

The theme of JavaOne 2011, held October 2–6, was "Moving Java Forward"—and by the end of the conference, most attendees were feeling a new momentum reminiscent of Java's initial rush into prominence more than a decade ago. What Oracle's stewardship of the Java platform means was brought into clarity at this year's JavaOne. The new direction is coherent and focused, eschewing disjointed efforts that are tangential to the prime objectives, ultimately leading to a cohesive Java platform where all the pieces fit together synchronously. Big topics at the show included Java/JDK 8; Java Platform, Enterprise Edition (Java EE), Java Platform, Standard Edition (Java SE), and Java Platform, Micro Edition (Java ME); and the Java Community Process (JCP), Java user groups, and of course, the Java community. Here are some highlights:

PHOTOGRAPHY BY HARTMANN STUDIOS

Cameron Purdy

Adam Messinger

Java/JDK 8 Roadmap

JavaOne provided an opportunity to hear about Oracle's long-term vision for investment and innovation in Java. At the Java strategy keynote, Oracle's **Hasan Rizvi**, **Adam Messinger**, and **Cameron Purdy** laid out a clear roadmap to Java 8. Responding to feedback from the community, the Oracle Java team decided that Java will return to a two-year major release cycle (which worked well in the early 2000s). The release of Java 8 will be delayed until the summer of 2013, but its scope is now expanded.

Java 8 will be a complete consolidation of the Java platform, including Oracle JRockit feature integration into the OpenJDK, Project Jigsaw (modularity), Project Lambda (closures), JavaFX 3.0, enhanced JavaScript interop, and more, all fully supported on Windows, Mac OS X, Linux, Oracle Solaris, and embedded platforms, with added support of new devices as well. NetBeans.next will provide support for all the new features.

Java EE, Java ME

The focus for Java EE 7 development will be multitenancy, capacity on demand, and autoprovisioning, looking ahead to a future standardization of the cloud platform. The Java ME focus is convergence of Java ME, Java SE, and Java SE for Embedded Devices, leading to a coherent micro/embedded Java platform.

JCP, Java User Groups, Community

A new commitment to community involvement and openness was also apparent at JavaOne 2011. The JCP has been re-energized by the presence of Java user groups (JUGs) on the JCP Executive Committee and the approval of JCP.next (JSR-348). The JUG-led initiative "Adopt-a-JSR" seeks to expand community involvement in determining the future of the Java platform by promoting active JUG participation in individual JSRs.

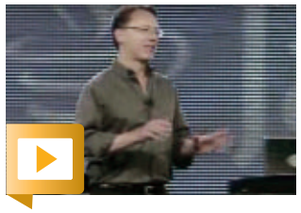
Twitter Joins the Party

In an announcement that surprised everyone, Twitter's Rob Benson—a prominent Java stakeholder with a vested interest in a highly performant Java platform—said that his company would join not only the OpenJDK Project but the JCP as well. Read Twitter's formal announcement here.

Forward, Ho!

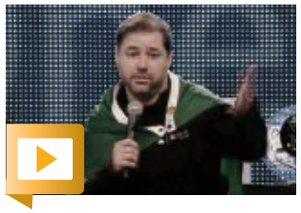
In conclusion, JavaOne 2011 was a momentous event. On all fronts, Java is indeed moving forward.

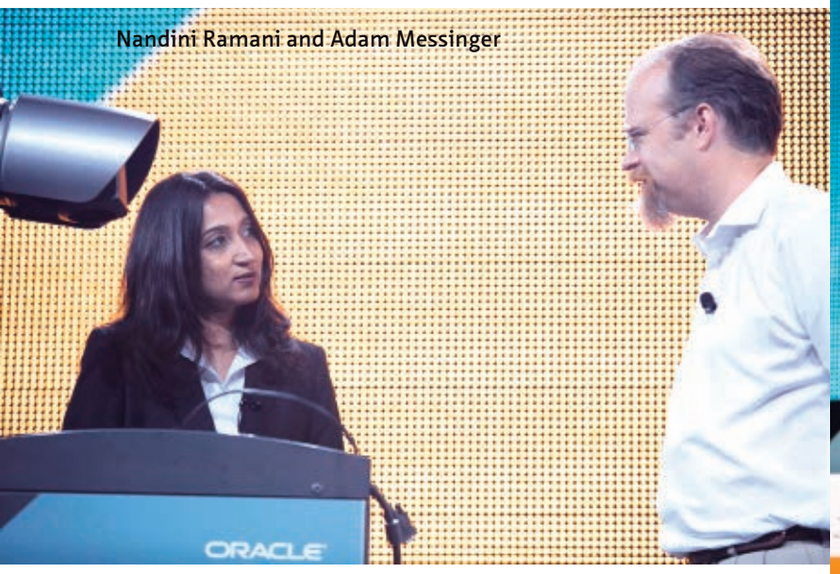
JavaOne Technical Keynote



Java Strategy Keynote

Java Community Keynote





Nandini Ramani and Adam Messinger

Richard Bair

JavaFX 2.0 Is Here!

The big news at JavaOne 2011 was the release of JavaFX 2.0, an advanced Java UI platform for enterprise business applications—and the next step in the evolution of Java as a premier rich-client platform.

At the JavaOne technical keynote on Monday, October 4, **Richard Bair**, chief architect, client Java platform, at Oracle, detailed the general availability of JavaFX 2.0 for Windows (32-bit XP, 32- and 64-bit Windows Vista, and Windows 7). He also announced NetBeans 7.1 Beta (featuring full JavaFX 2.0 support); the JavaFX 2.0 developer preview for Mac OS X; and JavaFX Scene Builder early access.

Key features in JavaFX 2.0 include 100 percent Java APIs, the new FXML UI markup language, and full Swing integration. JavaFX 2.0 provides a Web component based on WebKit, which allows developers to seamlessly mix and match native Java capabilities and the dynamic capabilities of Web technologies.

PHOTOGRAPHY BY HARTMANN STUDIOS

At the Java strategy keynote on Tuesday, Oracle Vice President of Development **Adam Messinger** announced that Oracle is submitting a proposal to open source the JavaFX platform as a new project within the OpenJDK community. Oracle will be working with the JCP to propose JavaFX as an official standard part of the Java platform, Messinger said. Oracle intends to initially contribute the JavaFX UI controls and related libraries; other JavaFX components are planned to follow in multiple phases.

Nandini Ramani, vice president of development, Java client group, at Oracle, generated a lot of buzz at that same keynote with a demo of JavaFX 2.0 running on an Apple iPad 2.0, a Samsung Galaxy Tab 10.1 running Android 3.1, and an Acer Windows 7 tablet. "We want to hear from the community," Ramani said. "If this is something you want to see, we're happy to make it a priority." (Read more in the Nandini Ramani interview, "Shock the Senses," on page 32.)

Looking to the future, full Mac OS X support will coincide with the Java 7 for Mac release, which will be followed by support for additional platforms (Linux, Solaris, and so on). You can listen to Nicolas Lorain's JavaOne 2011 session "Introduction to JavaFX 2.0" and other JavaFX sessions at parleys.com.

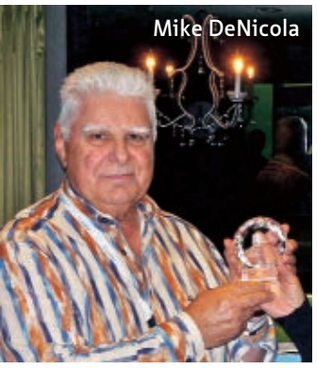
DUCHESSES SHARE THEIR VIEWS ON IT

The Duchess network connects women Java developers from around the world. At JavaOne 2011, Regina ten Bruggencate, current president, and Clara Ko, cofounder, discussed the role of women in IT in a birds-of-a-feather session.

Women engineers in the Duchess network connect at local Java user group (JUG) meetings and globally at conferences and online. Started in the Netherlands, the network saw membership double in the last year, thanks to very active Duchesses and supportive JUG leaders and members. The network now has 400 members in 57 countries.

Duchess promotes "changing the image of the nerdy guy as being the typical IT professional," ten Bruggencate says. "The industry is moving toward having developers that are willing to and good at collaborating to make better software," echoes Ko. Join them online and meet with them at Devoxx.


Duchess President Regina ten Bruggencate



JAVA COMMUNITY PROCESS AWARD **WINNERS ANNOUNCED**

The winners of the **Ninth Annual Java Community Process (JCP) Awards** were announced at a JCP ceremony that took place at JavaOne. There are three award categories: JCP Member/Participant of the Year, Most Innovative JSR, and Outstanding Spec Lead.

Mike DeNicola of Fujitsu won the JCP Member of the Year Award, for his contributions as chair of the JCP.next JSR-348 working group. DeNicola's efforts on the JCP reform plans increased the pace of progress on this critical JSR.

JSR-292, "Supporting Dynamically Typed Languages on the Java Platform," won the Most Innovative JSR Award. The awards committee noted, "As the first JSR specifically designed to support languages other than Java, JSR-292 will ensure the long-term success of the Java VM."

The Outstanding Spec Lead Award went to JSR-292's **John Rose**, consulting engineer at Oracle, for his efforts in forging consensus among the diverse participants in JSR-292, including the Expert Group and the wider Java Virtual Machine language community. Following the awards, Rose expressed being deeply touched by the recognition, and thanked his colleagues. "Together we introduced fundamentally new instructions and datatypes into the Java Virtual Machine, an undertaking that was both ambitious and complex," he wrote. "Carrying out this work required the dedicated efforts of the JSR-292 Expert Group members, our fellow engineers, corporate sponsors, and the Da Vinci Machine community."

There could only be three winners, but all of this year's nominees deserve recognition for their significant contributions to the JCP.

Java 8 Lambda Expressions Syntax Decided by JSR-335 Expert Group

In early September, the Expert Group for JSR-335 ("Lambda Expressions for the Java Programming Language") made an important decision regarding syntax. Java Lambda expressions will consist of

[parameters], [a single arrow `->`], and [an expression or statement(s)].

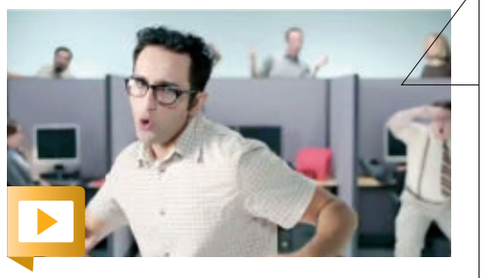
Here are some examples:

```
x -> x + 1
(x, y) -> x + y
(x, y) -> { System.out.printf("%d + %d = %d%n", x, y, x+y); }
```

The syntax is similar to the lambdas in C# and Scala, with the exception that those languages use `=>` as their arrow. The Java Lambda Expert Group decided that a single arrow `->` will provide greater clarity, for example, in Java expressions that also include `==` and `<=` operators.

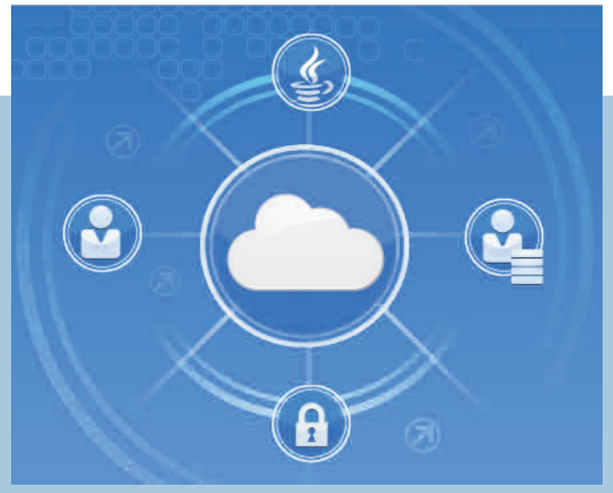
Reviewing the Java Lambda syntax decision, Specification Lead Brian Goetz commented: "Despite extensive searching, there was no clear winner among the alternatives (each form had some good aspects and some really not very good aspects, and there was no form that was clearly better than the others). So, we felt that it was better to choose something that has already been shown to work well in the two languages that are most like Java—C# and Scala—rather than to invent something new."

For background on closures in Java, see Mark Reinhold's blog post "Closures for Java." Soon you'll be able to experiment with Lambda expressions by downloading JDK 8. Visit the OpenJDK's Project Lambda site for more information.



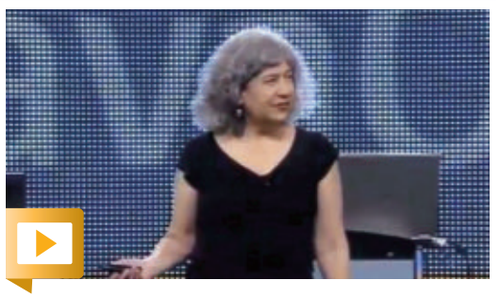
LIVING THE JAVA LIFE

Do you code hard in your cubicle? This original rap music video, made for JavaOne 2011, celebrates the "Java Life." Watch and see if *you're* living the Java life.



Oracle Public Cloud Provides Java Enterprise Interoperability and Scalability

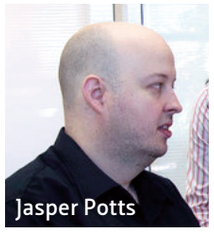
"Write once, run anywhere" doesn't have to exclude the cloud. Oracle's new Java-centric infrastructure service, **Oracle Public Cloud**, "is based on industry standards and supports full interoperability with other clouds and your data center." A look at the specs reveals what this means. The foundation of Oracle's cloud is clustered Oracle WebLogic Server 11*g* application servers running on the Java Platform, Enterprise Edition 5 (Java EE 5) stack. The toolkit suite includes Servlet 2.5, JSP 2.1, JSF 2.0, Enterprise JavaBeans 2.1 and 3.0, JPA 2.0, JAX-WS 2.1, and JAX-RS 1.1 (Jersey 1.9). The architecture is single tenant, with triple mirroring and offsite tape backup, for greater assurance of data security and integrity. If you're running an enterprise Java applications suite and you need higher availability and elastic scalability, consider Oracle Public Cloud. The service is currently available in limited trial mode.



Linda DeMichiel, Java EE 7 specialization lead at Oracle, talked about moving Java EE into the cloud at the Java EE 7 Platform keynote.

Jason McGee, chief architect, cloud computing, at IBM talked about the challenges of running Java applications on the cloud at the IBM keynote.

JavaFX Interfaces with the Xbox Kinect



What can you do with an Xbox Kinect and JavaFX 2.0? Attendees at Oracle Java Evangelist Simon Ritter's JavaOne session "Interfacing with the Interface: JavaFX 2.0, Wiimote, Kinect, and More" experienced a hint of what's possible when Developer Experience Architect for Java Client Jasper Potts took to the stage for a session-ending demo. The Kinect has an array of 3-D visualization, audio, and motion sensors (useful for playing Xbox games). But, it also includes a C++ SDK that provides programmers with access to data from the sensors.

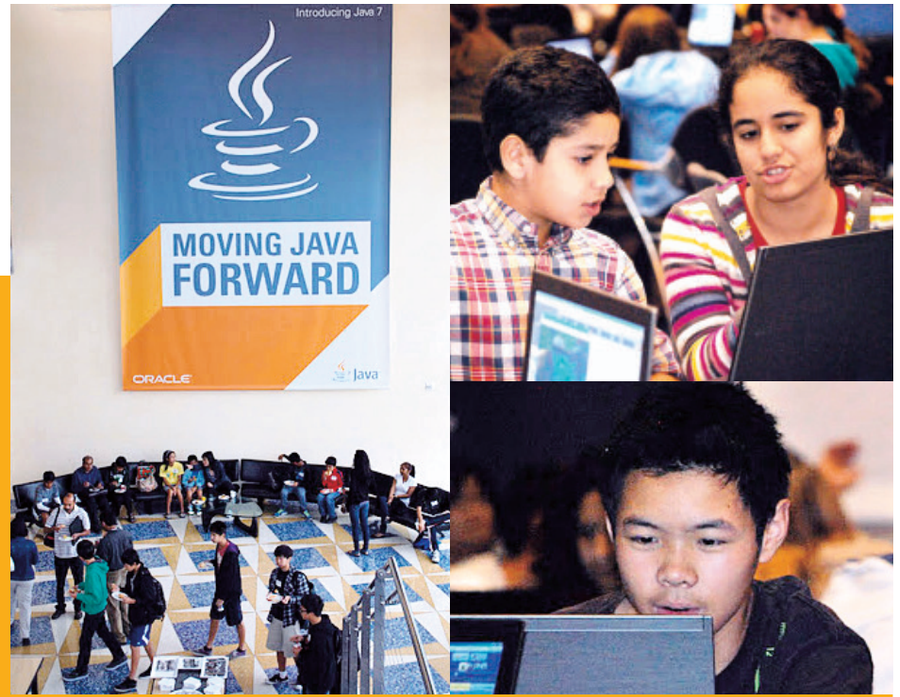
Ritter and Potts used the OpenNI framework to wrap the Kinect API so it could be accessed by JavaFX using Java Native Interface. It took some work, and some tweaks, but ultimately they were able to access 3-D depth data from the Kinect sensors using this type of JavaFX code:

```
context =
  Context.createFromXmlFile(config.getKinectXMLConfig(),
  scriptNode);
depthGenerator = DepthGenerator.create(context);
DepthMetaData depthMetaData = getDepthGenerator().getMetaData();
userGenerator = UserGenerator.create(context);
width = depthMetaData.getFullXRes();
height = depthMetaData.getFullYRes();
```

Considering the demo and the JavaFX technology behind it, Potts said, "This is a cool 3-D application from JavaFX labs. We showed Duke in 3-D and I got to pretend to be Duke, with the Kinect controller controlling it. It was a lot of fun and all went well onstage, so I'm really pleased."

As Ritter illustrated at the start of his presentation, humans have been interfacing with machines for a very long time (for example, typewriters were a great invention). Today software developers can apply open source technologies such as JavaFX to interface with modern instrumentation devices (like the Kinect). As Ritter said, the thing to do now is "be inspired, go out there, and build more stuff!"

PHOTOGRAPH BY BOB ADLER



Moving Java Forward: Summer Workshop 2011

More than 300 high school students, children of Oracle employees, parents, and teachers attended the five-day Java Summer Workshop, August 8–12, 2011, at Oracle headquarters in Redwood Shores, California. Students created their own animations with Alice and Greenfoot, two award-winning software programs that have been used for years to visually teach Java programming basics. These kids were buzzing with creativity—take a look at the winning Alice animations.

The Java Summer Workshop focused on building confidence and interest in Java and computer science in high school students at a crucial time in their career planning. This investment in the next generation ensures that new talent will enrich the Java ecosystem.

Alice and Greenfoot are free to download, and free workshop training is available on Oracle Technology Network.

PHOTOGRAPHY BY DAN KILDAHL

Hear from the next generation of Java programmers.

EVENTS

JAVAONE 2011 LATIN AMERICA *DECEMBER 6–8, SÃO PAULO, BRAZIL*

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

NOVEMBER

QCon International Software Development Conference
NOVEMBER 14–18, SAN FRANCISCO, CALIFORNIA
This practitioner-driven conference is designed for team leads, architects, and project managers. It includes two tutorial days and three conference days with 18 tracks and more than 80 speakers covering relevant and emerging topics in software development. Don't miss the featured track, Why Java Is Still Sexy.

Devoxx 2011
NOVEMBER 14–18, ANTWERP, BELGIUM
This annual European Java conference is organized by the Belgian Java User Group. It features a two-day in-depth Devoxx University program, followed by three days of conference sessions. Demos of Java tools, hands-on labs, birds-of-a-feather sessions, and opportunities for attendees to give quick presentations are also on the agenda.

DECEMBER

IndicThreads Conference on Java
DECEMBER 2–3, PUNE, INDIA
The 6th Annual IndicThreads Conference on Java is the premier independent conference on Java technology in India and is *the* place to be to learn the latest in the Java world while meeting with like-minded individuals from across the industry. The conference focuses on new and groundbreaking technologies and emerging trends, successful practices, and real-world learning. Session topics include Java language specifications and standards; enterprise Java; Java for mobile devices; Java for multicore computing; optimization, scaling, caching, and performance tuning; and more.

PHOTOGRAPH BY GETTY IMAGES

OSCON Java

The 13th annual O'Reilly Open Source Convention (OSCON), held July 25–27 in Portland, Oregon, was colocated with the inaugural OSCON Java and OSCON Data conferences. Explaining the addition of OSCON Java, Edd Dumbill, OSCON cochair, said that the Java platform is the most stable, complete, and scalable technology for innovations in cloud computing and big data. "You see start-ups beginning with scripting languages and porting bits of their architecture over to Java when they need scaling," he added.

OSCON Java keynotes included "Open Source, Java, and Oracle—Cracking the Code" with Steven G. Harris, former Oracle senior vice president; "JDK 7 in a Nutshell" with Joe Darcy, Project Coin lead engineer; and "Who Needs Standards?" with Patrick Curran, Java Community Process chair.

Sonatype
Brings Java.net Projects into the Central Repository

It's natural for open source projects to build upon the work done by other open source projects. Sonatype is facilitating the availability of components from Java.net's large open source project base, by bringing Java.net project artifacts into the Central Repository, a leading source for open source Java components. Sonatype's **Nexus Maven service** was made available to Java.net projects in May. All Java.net projects, regardless of size, have access to the new Maven service. The benefit for the global Java open source community is that Java.net project owners can now automate and control synchronization of their Java.net project artifacts to the Central Repository. As a result, any Maven project can now leverage Java.net project assets more easily to deliver applications faster, at a higher quality, and with less risk.

"Developers now have access to Java.net components directly from the Central Repository, requiring no debugging or additional configurations," says Jason van Zyl, Sonatype founder and CTO.

PHOTOGRAPHY BY PINAR OZGER

JAVA BOOKS

***JAVA, A BEGINNER'S GUIDE*, FIFTH EDITION**
By Herb Schildt
Oracle Press (September 2011)

Learn the fundamentals of Java programming in no time from best-selling programming author Herb Schildt. Fully updated to cover Java Platform, Standard Edition 7, *Java, A Beginner's Guide,* Fifth Edition starts with the basics, such as how to compile and run a Java program, and then discusses the keywords, syntax, and constructs that form the core of Java. You'll also find coverage of some of Java's most advanced features, including multithreaded programming and generics. Get started programming in Java right away with help from this fast-paced tutorial.

Java Magazine subscribers will receive a 40 percent discount.

Read a sample chapter: "Java Fundamentals."

***JAVA 7 : THE COMPLETE REFERENCE*, 8TH EDITION**
By Herb Schildt
Oracle Press (June 2011)

Herb Schildt's classic Java programming resource is thoroughly updated to cover the new features in JDK 7. This expert guide offers comprehensive coverage of the Java language, its syntax, keywords, and fundamental programming principles. Descriptions of key API libraries are also included. Of course, details on new JDK 7 features, such as try-with-resources, type inference via the diamond operator, catching multiple exception types (multicatch), upgrades to NIO, and the fork/join framework, are provided.

JavaFX 2.0

Green Light

Startup **dooApp** supports the work of green building professionals with an innovative testing tool. **BY PHILIP J. GILL**

Until recently, architects, engineers, and construction professionals have concerned themselves with making buildings safe, structurally sound, functionally appropriate, and, of course, aesthetically pleasing. In the past few years, as energy costs have risen and concerns about the environmental impact of greenhouse gases have grown, those same professionals and others have realized that they also need to make buildings more resource efficient and environmentally friendly.

Known as *green building*, *green architecture*, or *sustainable design*, this worldwide movement is changing the way the buildings in which we live, work, and play are

Christophe Dufour (left), Antoine Mischler (second from left), and members of the dooApp team test drive the latest release of their Infiltrea application.

PHOTOGRAPHY BY ERIC FLOGNY/GETTY IMAGES
ART BY PAULINA MCFARLAND

SNAPSHOT

DOOAPP
dooapp.com/fr
Headquarters:
Lille, France
Industry:
Software
Employees:
7
Java version used:
Java Platform, Standard Edition 6

designed and constructed. It is also opening opportunities for innovative software developers to build new tools to address the specific needs of green building professionals. One such company is dooApp, a Java startup based in Lille, France.

"Our software makes the work of the green building professional easier," says Antoine Mischler, who cofounded the firm two years ago with Christophe Dufour and now serves, with Dufour, as co-managing director. "Our first application, Infiltrea, is dedicated to professionals who measure the airtightness of buildings," explains Mischler.

Because Mischler and Dufour had both been Java developers in the financial software market, Java was the natural choice as the development platform for Infiltrea. "We knew that it would make it easier to recruit people with good Java skills and that we would have a lot of existing libraries we could reuse, which would make the whole development process quicker. We also knew we would have a lot of tools available to us to manage and test the development cycle," says Mischler. "We had had a great experience with Java before, and we wanted to take that technology further."

Infiltrea, built on Java Platform, Standard Edition 6 (Java SE 6), makes extensive

To the Web and Beyond with 2.0

Infiltrea has been on the market for about a year now and is marketed in France with dooApp partner Testoon, a Paris-based distributor of test and measurement equipment. Testoon packages Infiltrea with its blower door systems.

With the recent introduction of JavaFX 2.0, Lille, France–based software startup dooApp has rewritten Infiltrea to take advantage of the framework's new features, says dooApp Cofounder and Co-managing Director Antoine Mischler. These include improved binding and better debugger and cascading style sheet (CSS) tools. The most important feature, however, may well be JavaFX's new support for Web applications through its WebView component.

In the 2.0 release of Infiltrea, dooApp is Web-enabling its application using the JavaFX WebView component. In earlier JavaFX frameworks, applications had to run on a local desktop device, such as a PC or Mac. Now, WebView is an embedded browser for JavaFX 2.0 applications whose features include the ability to render HTML content from local and remote sources, to execute JavaScript commands, and to apply effects and animations.

At the same time, dooApp has partnered with a regional association of green building professionals to develop a specialized social network for green building diagnostic and measurement professionals. The social network will allow Infiltrea users to share tips, techniques, and experiences and build a common community of knowledge across the internet.

"The idea is to integrate the network directly into Infiltrea using the new WebView component introduced in JavaFX 2.0," says Mischler. "This allows a very tight integration between desktop and Web applications, in this case Infiltrea and the social network. I think this is one of the most innovative features in JavaFX 2.0 and will change the way desktop and Web applications interact."

Mischler says dooApp is already looking at other applications for green building professionals. Linking up with green building professionals of all kinds through the social network will help drive the development of future applications. In this way, he hopes, dooApp and Java can change the way green building professionals do their work, just as green building technologies are changing our built environment.

Meet Vinny Senger, the technical lead of jHome, a complete home automation API based on GlassFish Server Open Source Edition and Java Platform, Enterprise Edition that lets you control anything in your home.

Duke's Best and Brightest

The Duke's Choice Awards were created in 2003 to honor extreme innovation in the world of Java technology. The 2011 Duke's Choice Awards, presented at JavaOne 2011, showcased the amazing results of projects supported by thousands of development hours. The winning projects and companies are as follows:

- **Arquillian.** This project simplifies integration testing for Java-based applications.
- **dooApp.** Infiltrea is dooApp's JavaFX and Java Platform, Standard Edition solution, designed for green building professionals whose job involves measuring the airtightness of buildings.
- **Inductive Automation.** Ignition is Inductive Automation's Java-based Web application that integrates different manufacturing machines using a central Web server.
- **jHome.** This complete home automation open source API, based on GlassFish Server Open Source Edition and Java Platform, Enterprise Edition (Java EE), enables developers to control anything in their homes.
- **JFrog.** JFrog's Artifactory is the world's first binary repository manager and is built with the Content Repository API for Java specification.
- **JRebel.** This Java Virtual Machine plug-in enables Java developers to instantly see any code change made to an application.
- **LMAX Disruptor.** This solution is a multithreaded, open source concurrent programming framework designed for high-performance and low-latency transaction processing.
- **Rockwell Automation.** This company's Java-enabled human–machine interface line of products will allow the automated communication and exchange of data to factory floor lines.
- **Sodbeans Project.** This open source, NetBeans-based module suite is designed to enhance accessibility for the blind in modern programming environments.
- **The Netty Project.** This Java-based, new I/O client/server framework enables quick and easy development of network applications.

"The Duke's Choice Awards showcase the incredible creativity that takes place every day in the Java community," says Amit Zavery, vice president of product management in Oracle Fusion Middleware. "This year's winners demonstrate the countless ways that Java technology can be used to create innovative applications and products. Put in the hands of creative and passionate individuals, Java can have a significant impact on our lives." —*Marta Bright*

use of JavaFX, a leading user interface, graphics, and media framework for developing rich internet applications in the Java environment (see sidebar, page 11). The software is also the winner of this year's Duke's Choice Award for Innovative Green Technology, presented at this year's JavaOne conference in San Francisco, California, in October.

AIRTIGHT, ENERGY EFFICIENT

Airtightness is one of the key measurements of a building's energy efficiency, and it's a major goal of green building. According to the U.S. Energy Information Administration, buildings in the U.S. create 48 percent of all domestic greenhouse gas emissions and consume most of the electricity the nation's power plants generate. Making buildings more energy efficient reduces both the demand and pollution.

When buildings "leak," some of the energy their heating and cooling systems consume to create a comfortable indoor environment is wasted; buildings that are airtight, on the other hand, consume less energy because they waste less through leakage. "Green buildings need to be airtight," explains Mischler.

In Europe, regulation EN 13829 mandates what is called a *blower door test* to determine if new construction, both residential and commercial, meets airtightness requirements. It also defines the testing method, criteria, and reporting requirements. Individual countries have their own regulations; in France, for example, RT 2012 and its stricter airtightness standard began taking effect in late 2011.

Infiltrea provides an end-to-end solution for the airtightness testing process, says Mischler. The application includes a blower door connection, standard tests and charts, and report-writing features. The charts, created with JavaFX, provide the main view of the test's measurements.

dooApp's Infiltrea application helps green building professionals to measure airtightness, a critical component in green building. The 2.0 release of Infiltrea is Web-enabled using the JavaFX WebView component, as seen here in the project outline view.

Antoine Mischler (center) and the dooApp team brainstorm about enhancements to the Infiltrea application.

"The goal of the process is to compute some indicators saying whether the building is good or bad," says Mischler, noting that *good* means it meets regulatory requirements.

A blower door system consists of a calibrated fan, a door-panel system, and a device to measure the fan flow and building pressure. The blower door fan is temporarily sealed into an exterior doorway and then blows air into or out of the building, creating a pressure difference between inside and outside. The pressure difference forces air through all the holes and penetrations in the building's exterior envelope, including the walls, doors, and windows. The tighter the building, the fewer holes or leaks there are in its envelope and the less air that is needed from the blower door fan to create a change in the building's pressure.

FAST FACT

Buildings in the U.S. create 48 percent of all domestic greenhouse gas emissions.

The Infiltrea software runs on Windows, Mac OS, and Linux; it records the test and presents a visual representation of its progress in real time. "Our software is connected to the measuring device and to the blower door, and acquires the measurements," says Mischler. "With Infiltrea, the professional has a chart that shows the pressure in the building and the airflow in real time."

Once recorded, the measurements are published in two reports, directly from Infiltrea. The first, a technical report, is a two-pager that includes the building's features, the testing criteria, and a chart of results. The second is a longer and more complex legal report that conforms to EN 13829. It can include photos, logos, plans, thermal surveys, and other relevant data and can be published in a number of formats, including PDF, .docx, HTML, RTF, and others. "At the end," Mischler says, "the software gives a full report to the professional that conforms to the regulations."

Infiltrea doesn't show *where* a building is leaking, however, and that is the next step in the process of making a building airtight. If a building doesn't pass, then the diagnostic professional conducts smoke tests to identify leakage sites. "Once the leakage site is identified, the professional will seal it and run the airtightness tests again until all the leakage sites are identified," explains Mischler.

THE NATURAL CHOICE

Going with Java and JavaFX was a natural choice for dooApp's green building applications. Mischler says that when he and Dufour were working as Java developers, they real-

ized they wanted to apply their expertise to an area in which they both had a strong personal interest: green building.

"What we wanted to do was to bring our technical skills to develop innovative software for professionals in the green building sector," says Mischler. "What we had seen is that most software developers in this sector were engineers or architects who had moved into software but lacked specific skills in software. And we thought that, as a result, the tools they had developed were not so innovative or user friendly."

Making the software interface user friendly was a high priority, so they chose JavaFX as Infiltrea's graphics and user interface framework, says Mischler. "We already had worked a bit with Adobe Flash, but we knew this was not well suited to design a professional desktop application because of performance and maintenance issues," Mischler explains. "Since [Microsoft] Silverlight is pretty much the same, we didn't look at it any closer."

"We both had experience with Swing, and when we started developing Infiltrea we had the choice between Swing and JavaFX," continues Mischler. "Swing is the 'old' Java graphics framework. So Swing was a mature and stable technology but had some major drawbacks: you have to write a lot of code for small results, it's hard to style components, and the default look and feel looks a bit outdated."

On the other hand, he continues, JavaFX "was a new technology that addressed these issues and introduced tons of cool new features, such as special effects and animations, as well as CSS skinning."

The choice of JavaFX wasn't made without trepidation, however. "Since it was a new technology, making the choice of JavaFX was somewhat of a gamble," Mischler concedes. "After considering both options, we made the choice to work with JavaFX and we do not regret this today. The technology is fun and is evolving in the right way." `</article>`

DUKE'S GREEN CHOICE

French startup dooApp is the winner of this year's Duke's Choice Award for Innovative Green Technology.

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

Rainer Schügerl, Director of Software Development and Security, SVC

Austria's E-Health System: DRIVEN BY JAVA

Smartcard system connects patients, providers, hospitals, and pharmacies throughout Europe. **BY DAVID BAUM**

PHOTOGRAPHY BY HEINZ S. TESAREK/GETTY IMAGES

Austria is well known for its advanced and generous social services—marked by excellent healthcare; a strong social security system; and an extensive network of hospitals, physicians, and pharmacies. Austrian Social Security Law, a public insurance system that includes 22 institutions offering a variety of insurance coverage and social programs, insures most Austrian citizens, and in recent years this national health system has been bolstered by one of the world's most advanced programs for smartcards and electronic health records.

Java-powered medical practice units (MPUs) and smartcard readers enable healthcare providers to read patient records and transmit health records to the SVC data center.

Today all insured citizens get a smartcard that verifies their insured status and facilitates the creation, transmission, and storage of electronic health records. The e-card system also authorizes citizens to avail themselves of a variety of e-health services, from preventive checkups to disease management programs. Thousands of healthcare providers have installed special equipment to scan the smartcards and transmit secure data across a secure health information network that spans the Austrian nation and extends to many other European countries within the scope of the NETC@RDS project of the European Union. Java is the *lingua franca* of the system.

"We chose Java because of the platform advantages, especially its tremendous portability among CPUs and hardware platforms," says Rainer Schügerl, director of software development and security at SVC, a public entity based in Vienna, Austria, that develops innovative solutions in the field of health telematics and e-government. SVC served as the systems integrator for the e-card project.

"Java provides a stable, high-quality programming language that suits all of our needs," Schügerl adds. "For enterprise-caliber development requiring high availability, reliability, and security, most Austrian organizations use Java."

SVC has made a giant step forward in improving the quality of medical care for citizens throughout Austria and beyond. By implementing the e-card infrastructure, the agency replaced a cumbersome system of paper health insurance vouchers that had dominated Austria's medical world for 50 years. Manual processes between insured people, employers, doctors, hospitals, and the Main Association of Austrian Social Security Institutions have been systematically replaced by electronic solutions and e-business procedures governing the transmission, storage, processing, and virtualization of health and administrative data.

SECURE INFRASTRUCTURE

The Main Association of Austrian Social Security Institutions is a group of statutory insurance providers that is responsible for the country's health, pension, and accident insurance. The association enlisted SVC to design and deploy the e-card system and an associated virtual private network. As a 100 percent subsidiary company of the Main Association of Austrian Social Security Institutions, SVC has a proven track record of enforcing transparency and security in the processing of health and administrative data, whether for clinical use, administration, or science. Its mission is to improve the efficiency of information processes among patients, health service providers, and health administrators.

PERFECT FIT

Java applets carry essential security features and are ideal for smartcards.

Schügerl has 40 software engineers on his team, including 20 Java developers. Due to the sensitive nature of health and medical information, the team needed a programming language with a robust security architecture. They decided to use Java Standard Edition for Embedded Devices, which supports the same platforms and functionality as Java Platform, Standard Edition (Java SE), along with additional capabilities for the embedded market such as support for small-footprint Java runtime environments (JREs), headless configurations, and memory optimizations. Java SE is ubiquitous on servers and desktop computers, thanks to its high-performance virtual machine and exceptional graphics support. To top it off, it has an extremely lightweight deployment architecture and a complete set of features and libraries for programmers.

Java applets are ideal for smartcards and similar small-memory devices because they carry all the essential security features such as cryptography, authentication, authorization, and public key infrastructure. For SVC, these features ensure that new e-card applications can securely access centralized resources while protecting critical data from theft, loss, and corruption. The Java API supports a wide range of cryptographic services including digital signatures, message digests, ciphers, message authentication codes, key generators, and key factories, allowing SVC developers to easily integrate security into their application code.

The smartcard does not contain software functions. Only the cardholder's administrative data is stored on the e-card—name,

Rainer Schügerl (second from right) takes a break with members of the SVC engineering team. In just seven months, SVC rolled out 8.5 million e-cards to Austrian citizens.

SNAPSHOT

SVC
www.svc.co.at/english

Headquarters:
Vienna, Austria

Industry:
Public services

Employees:
100

Java version used:
Java Standard Edition for Embedded Devices 6

academic title, insurance number, serial number, gender, and user group identification—along with certificate and identity link parameters. "The e-card is used primarily as an identification tool for patients and the key to their data," explains Schügerl. "In order to get paid, doctors insert their patients' e-cards into a smartcard reader. The administrative patient data is then cross-checked with a central social security database, which resides in a secure data center."

Only authorized medical practitioners can read the cards and the associated health records, and they must have special equipment called medical practice units (MPUs) to do so. Commonly known as *GINA boxes*, each MPU is a special-purpose computer that hosts a lightweight Java program to obtain data from the smartcard. The MPUs contain the application software that initiates secure business procedures. They interface with another piece of equipment called a LAN chip card reader to transmit patient data to the SVC data center over a virtual private health information network.

RAPID ROLLOUT

In a span of seven months, SVC rolled out 8.5 million e-cards to insured citizens, along with GINA boxes to 11,000 providers and medical practitioners. SVC continues to ship about a million e-cards every year and to bring new medical practices and health service providers on board. More than 12,500 healthcare providers now have the equipment and have been trained in the business procedures of the e-card system.

For the cardholder, medical treatment is now accessible without administrative barriers and without paper documents. Physicians can verify if a person is insured and which health insurance institution will pay for medical treatment. The system verifies insurance coverage for all citizens, even when they are traveling to member states of the European Union, the European Economic Area, and Switzerland. "The card allows them to receive medical treatment in these member states, for free or at a reduced cost. The intention is to allow people to continue their stay in a country without having to return home for medical care, or to take a job in a neighboring country and keep their health benefits," Schügerl explains.

SMART CHOICE

SVC chose Java because of the platform advantages, especially its tremendous portability among CPUs and hardware platforms.

FUTURE APPS

The e-card system has been designed to support a number of future projects, including electronic signatures for pharmaceutical prescriptions, emergency data, electronic vaccination records, and disease management programs. For example, a new e-medication system makes it easier for doctors and pharmacists to share information about potentially harmful interactions among medications.

Launched as a pilot project earlier this year in three regions of Austria, this service permits pharmacists, physicians, and hospitals to see a list of each patient's current medications. More than 100 physicians, about 50 pharmacies, and 6 hospitals participate in this project. The service will be fully implemented during 2012.

"Our research shows that people 70 years and older take up to 10 medications per day," says Schügerl. "Tracking and comparing pharmaceutical prescriptions with health records helps them to reduce negative interactions. So far about 7,000 citizens have participated in this pilot program." Within the scope of the pilot project, the participation of patients, physicians, pharmacists, and hospitals is voluntary.

After nationwide rollout of the e-medication system, citizens can "opt out" if they decide not to share their health information online—as some people prefer, due to privacy concerns. Because of the large number of involved stakeholders, it was not easy to balance the different interests and get the project running. But in general, both patients and providers have embraced these existing and emerging e-health applications, partly as a result of the robust, Java-based security architecture. Data is not stored on the card; it is housed in a secure data center. Only authorized providers and administrators can access these records, and they must have special-purpose equipment.

GLOBAL EXAMPLE

Following the slogan, "Move the data, not the patient," SVC's Java-based infrastructure has reduced the cost of healthcare administration, improved collaboration between healthcare professionals, and enabled higher-quality services for patients. Other Java applications communicate with the Main Association of Austrian Social Security Institutions to submit medical claims, verify pharmaceutical prescriptions, route disability reports, and transmit medical records between appropriate medical practitioners.

The e-card infrastructure has become the foundation of many future applications in the burgeoning health telematics sector, including a new "citizen card" that streamlines e-government services related to tax declarations, alimony payments, student loans, criminal records, residence registration confirmations, insurance data, and pension accounts. Many other countries have expressed interest in developing similar Java-based systems, including Australia, Bulgaria, Colombia, the Czech Republic, Finland, Germany, Iran, Japan, Kuwait, Lithuania, Macedonia, the Netherlands, Poland, Romania, Serbia, Slovenia, South Korea, Taiwan, Turkey, and Vietnam.

Java is the de facto standard for these smartcard initiatives, with 5 billion installations worldwide. Many technically savvy organizations come to the same conclusion as SVC: they depend on Java because it is a powerful and secure environment for applications that run on smartcards and other devices with very limited memory and processing capabilities. `</article>`

Stats on a Successful Rollout

SVC, a provider of health telematics and e-government solutions based in Vienna, Austria, has issued millions of e-cards to employed Austrian citizens, along with thousands of special-purpose computers that rely on Java to securely store and transmit patient data.

E-cards (active) in Austria	8,700,000
New e-cards issued	850,000 per year
Healthcare providers connected	12,500
Pharmacies connected	120
Hospitals connected	136
Care facilities connected	15
E-card transactions completed	667,763, 920
Average contacts per day	Approximately 500,000
All-time high (December 14, 2009)	629,150

Source: *SVC*

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

MICHAEL KÖLLING

BIO

Starting Your First Computer Game

Use Greenfoot to create your first interactive program.

In the Premiere issue of *Java Magazine*, I wrote an article about writing your first Java program using the Greenfoot environment. In that article, we made a turtle walk across the screen, either in straight lines or in circles.

Note: If you haven't done so already, go to that article to find the software you need to install to follow this project.

Getting an animated graphic onscreen on your first day is pretty good, but it gets boring after a while if you can't do anything more with it. So today, I'll tell you how to add keyboard interaction (that is, how to control your turtle's movement using keys), which might be a first step toward a little computer game.

Jumping Right In

When we left our turtle's act method last time, it looked like this:

```
public void act()
{
  move(4);
  turn(2);
}
```

We observed that this caused the turtle to run in a circle, because the act method is repeatedly executed when the program runs (see **Figure 1**).

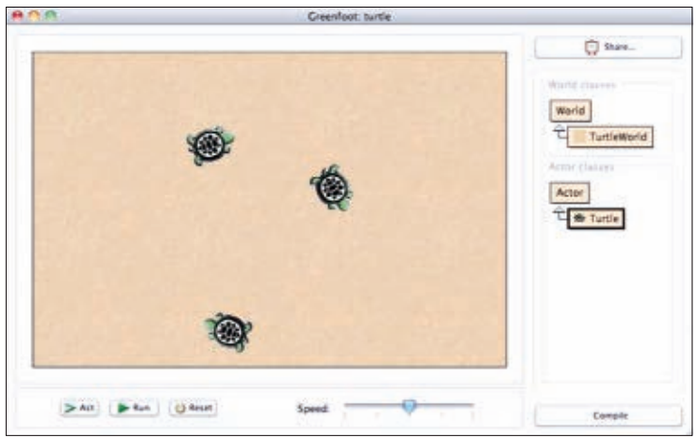


Figure 1

PHOTOGRAPH BY JOHN BLYTHE

Now, we want each turtle to run straight forward when we do not press any keys on our keyboard and to turn right or left if we press the right or left arrow keys.

To achieve this, we need two things:

- We need to be able to check whether a key has been pressed.
- We need a statement in Java that lets us state that we want to turn only if the right or left arrow key has been pressed.

ARROW CONTROL
You can now **control** a turtle on your screen with your **arrow keys**. Pretty good fun already!

Checking for Key Presses

Greenfoot has a method, called isKeyDown, that lets us check whether a given key on the keyboard is being held down. This method is in the Greenfoot class. We can, for instance, write the following to check whether the left arrow key is being pressed:

```
Greenfoot.isKeyDown("left")
```

The parameter (inside the parentheses) is the name of the key we want to check. The alphabetical keys have their character as their name (for instance, the A key is called a, the B key is called b, and so on). The special control keys have names indicating what they are; for example, left is the name of the left arrow key and—you guessed it—right is the name of the right arrow key.

The isKeyDown method, when called, will answer with a value of either true (yes, the left arrow key is being pressed) or false (no, the key is not being pressed).

An if Statement

The second part we need to make our plan work is a statement that lets us execute an action (in our case, turning) only under certain conditions (when the left or right arrow key is pressed). This is called an if statement.

An if statement has the following general form:

```
if (condition) {
  statements;
}
```

When an if statement is executed, it checks the condition, and then it executes the statements between the curly brackets only

if the condition is true. (Incidentally, it might be good to introduce some more-professional terminology: the part of the if statement containing the statements—the curly brackets and everything in between—is called the *body* of the if statement.) The body may contain a whole sequence of statements if that's what we want.

If the condition is false, the body is just ignored, and execution continues after the if statement.

Turning on Key Press

With these two bits together, we can now do what we need to do. We need to write the following:

```
if (the-left-key-is-being-pressed) {
    turn-a-bit-to-the-left;
}
```

This is not real Java code (it's pseudocode). We can now fill in the pseudocode with real Java code:

```
if (Greenfoot.isKeyDown("left")) {
    turn(-3);
}
```

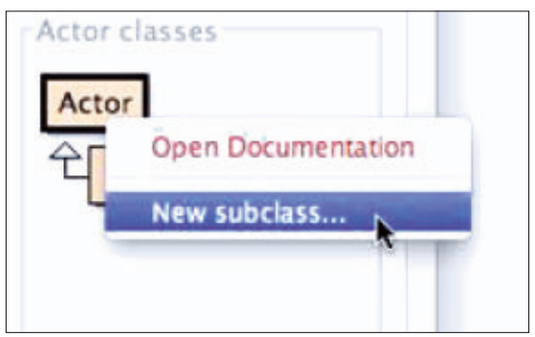


Figure 2

This code now states that we will turn minus 3 degrees during every act cycle in which the left arrow key is held down. (Note that we can use a negative angle to turn left.)

Doing the same for the right arrow key to turn right is now simple. We just write a similar statement again, this time turning 3 degrees for the right arrow key. The complete new act method is shown in **Listing 1**.

Try it out. Compile your classes, place a turtle into the world, and run the scenario.

You can now control a turtle on your screen with your arrow keys. Pretty good fun already!

Currently, the turtle will continuously walk forward. If you want to, you can change your code so that the turtle moves forward only when you press the up arrow. You do that by adding another if statement around the move(4) statement. The name of the up arrow key is up.

What Do Turtles Eat? Pizza!

Next, we want to give the turtle something to do: collect some food and eat it. To do this, we need some new objects in our world: pizza slices. (Yes, our turtle eats pizza. This is not a biology lesson—it's a computer game.)

Where do objects come from? From a class. So first, we need to create a new Pizza class.

Right-click the **Actor** class (see **Figure 2**) and select the **New subclass** function. In the resulting dialog box, enter Pizza as the class name, and search

LISTING 1 **LISTING 2**

```
public void act()
{
  move(4);

  if(Greenfoot.isKeyDown("left")) {
    turn(-3);
  }

  if(Greenfoot.isKeyDown("right")) {
    turn(3);
  }
}
```

See all listings as text

for the image of the pizza slice in the **food** category. Click **OK**.

As soon as the classes are compiled again, you can add a few pizza slices (as well as a turtle) to your world.

Try it out. Let your turtle run around to collect pizza. You will notice, though, that the turtle does not (yet) eat the pizza. It will just run straight across the pizza. We will fix that shortly.

Two Tips

If you worked along and experimented in Greenfoot for a bit, you will have noticed that you need to re-create all objects in the world after every compilation. This can get a little tedious after awhile, especially if we have many objects.

Here are two quick tips to make your life in Greenfoot a bit easier.

- You can shift-click into the world. This is a shortcut to create an object of the class that is currently selected in the class diagram on the right.
- When all your objects are in the world as you want them to be at the start of your game, you can right-click the world background and select **Save the World** to save this world state. This state will then be re-created automatically after every compile or reset.

DID YOU KNOW?
isKeyDown lets us check whether a given key is being held down.

Collision Detection

OK, we have pizza. We have a turtle. The turtle is hungry.

The last bit to do for today is to make the turtle notice when it runs into some pizza and to make the turtle eat the pizza. For this we need what's called *collision detection* to detect when we've run into a pizza slice.

The full code to do this is in **Listing 2**. This code should be added to the end of the

act method in the Turtle class. There are several new elements in this code segment. Let's look at them one by one.

The first line calls a Greenfoot method: getOneIntersectingObject(Pizza.class). This method checks whether the turtle intersects an object of class pizza (or, in other words, it asks, "Are we touching a pizza slice?"). If we are touching a pizza slice, this method returns the pizza object we found. If not, it returns a special value, null, which tells us that we are not touching anything.

TEDIUM CHECK

Re-creating all the objects after **every compilation** can get tedious.

The equal symbol in the first line is an *assignment* operator. It stores the result of our collision check (the pizza object or null) in a *variable*. A variable is a bit of storage where we can store objects.

In our case, the variable has been declared using the following statement at the beginning of the first line:

```
Actor pizza
```

This declares a variable named pizza that can hold Actor objects (objects that live in the Greenfoot world).

After the first line, the pizza variable holds the pizza object we ran into, or it holds null if we didn't run into a pizza slice in this step.

Next follows another example of an if statement. In pseudocode, the next lines state the following:

```
if(we-have-run-into-pizza) {
  remove-the-pizza-object-
from-our-world;
}
```

The condition check is done with the following expression:

```
if(pizza != null) {
```

Figure 3

The != symbol means *not equal*. So, here we are saying, "If our pizza variable is not equal to null..."

The effect is that the instruction in the body of the if statement is executed only if the pizza variable is not null, that is, if we have run into some pizza.

The next line then removes the pizza object from the world. **Listing 3** shows the complete version of our act method at this stage.

Try it out. Once you have written this code, your turtle can walk around and eat pizza (see **Figure 3**).

LISTING 3

```
public void act()
{
  move(4);

  if(Greenfoot.isKeyDown("left")) {
    turn(-3);
  }

  if(Greenfoot.isKeyDown("right")) {
    turn(3);
  }

  Actor pizza = getOneIntersectingObject(Pizza.class);
  if(pizza != null) {
    getWorld().removeObject(pizza);
  }
}
```

See all listings as text

Conclusion

In this article, you learned a bit more about writing Java code and about some useful Greenfoot methods.

- If statements can be used to execute an action only under a certain condition.
- If statements have a condition and a body. The body is executed only if the condition is true.
- The Greenfoot.isKeyDown method can be used to check for key presses.
- The Greenfoot.isKeyDown method can be used as the condition in an if statement.
- Collision detection can be used to check whether one object touches another.
- Variables are used to store objects.
- The special value null is used to indicate that we do not have an object. `</article>`

LEARN MORE

- The Greenfoot gallery
- Young developer resources
- "Wombat Object Basics (Young Developers Series, Part 1)"
- "Wombat Classes Basics (Young Developers Series, Part 2)"

Introduction to RESTful Web Services (Part 2)

Handle the JSON response format with an HTML5 Web client.

MAX BONBHEL

BIO

In Part 1 of this three-part series, I showed how to quickly create and deploy a Java Platform, Enterprise Edition (Java EE) application that uses Representational State Transfer (REST) Web services with NetBeans. The application developed in Part 1 will be the basis for this second article on implementing new functionalities.

Now, in Part 2, we will see in detail how to use JavaScript Object Notation (JSON) to handle the response returned to the client. We will also add a small dose of HTML5, JavaScript, and Ajax to call the Web services, store the information locally, and display the information later.

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

HOW COOL IS THAT?
HTML5 provides **very cool functionalities** for **storing** the data in your browser and **accessing** the data with JavaScript after the page is loaded.

JSON and HTML5

JSON is a human-readable data format based on JavaScript technology. The structure of JSON is easier to parse and extend than traditional XML. JSON format is used by most applications that provide Web services through Web clients.

HTML5 is the next generation of HTML. It provides new features that are necessary for modern Web applications. It also standardizes many features of commonly available browser technologies that Web developers have been using for years to create rich Web applications without the need for plug-ins.

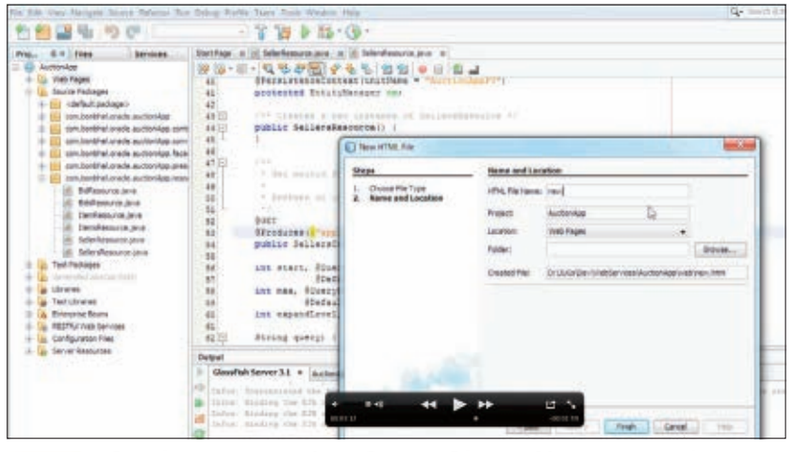
HTML5 is designed to be cross-platform, and it does not require a specific operating system. The only thing we need is a modern Web browser, such as the latest version of Firefox, Opera, Safari, or Chrome.

We will use the HTML5 storage capabilities to store information on the browser, retrieve it later, and finally display it. This article was tested with Firefox 6.0.2 and Google Chrome Release 14.0.835.186.

Let's Code and Test the JSON Representation

To begin, we will modify the resources to ensure that the server returns a response in the format that we want (JSON).

1. Create the method that will handle the Web services response for the seller:
   a. Select the **Source Package** node of the AuctionApp project and open the SellersResource.java file located in the package com.bonbhel.oracle.auctionApp.resource.
   b. Remove the XML MIME type (application/xml) from the @Produces annotation of the method get(). The class



See how to use JavaScript Object Notation with HTML5 to store, retrieve, and display information.

PHOTOGRAPH BY
ALLEN MCINNIS/GETTY IMAGES

should now look like **Listing 1**.

c. Select the **Source Package** node of the AuctionApp project and open the SellerResource.java file located in the package com.bonbhel.oracle.auctionApp.resource.

d. Remove the XML MIME type (application/xml) from the @Produces annotation of the method get() to handle the JSON response only. The class should now look like **Listing 2**.

e. Repeat Steps b, c, and d for each resource.

2. Test the JSON and XML response format:

a. Build and deploy the project.

b. To test the JSON response, open your browser and type the resource URL http://localhost:8080/AuctionApp/resources/sellers/.

The JSON representation of the resource (Sellers) will be displayed. You can see all the Sellers that you already created in JSON format. Save the result when prompted by your browser and open the saved file with any text editor to see the JSON string.

If you see the result shown in **Listing 3**, it means your RESTful services are working correctly. Now that the RESTful Web services are up and running, we are ready to consume them using any client.

Using an HTML5 Client

We need to use a Web client. So we are going to quickly build a sample HTML5 page front end to allow us to perform the following tasks using HTML, JavaScript, and Ajax functionalities:

- Call the Web services for the Seller.
- Store the Seller's information on the browser (HTML5 capability). The concept works like the cookies in your browser, but it's recommended for larger data.
- Retrieve the information stored in the browser.
- Finally, display the information.

JavaScript will be used to invoke the Web service via the XMLHttpRequest object provided by Ajax technology.

1. Create a sample HTML page:

a. Right-click the **Web Pages** node of the AuctionApp project and select **New**; then select **HTML**.

b. Type an HTML filename, such as JsonClient.

c. Type a folder name, such as jsonUI.

d. Click **Finish**.

A new resources HTML file is added to the project.

2. Edit the created page and modify the HTML <body> tag, as shown in **Listing 4**.

3. Add the JavaScript functionality to invoke the Web services and handle the data storage by modifying the HTML <head> tag, as shown in **Listing 5**.

4. Retrieve and display the stored information:

a. Modify the HTML <head> tag as shown in **Listing 6**.

b. Save the HTML file.

c. Build and deploy the project.

Testing the HTML Client Application

1. Call the Web services for seller #2:

a. Open your browser and type the resource URL http://localhost:8080/AuctionApp/jsonUI/JsonClient.html.

b. Enter the seller ID (2).

c. Click **Load seller in local browser**.

A confirmation message is displayed.

2. Display the stored information:

a. Retrieve the information for seller #2 from the local storage by clicking **Read Values**.

The first name and last name of seller #2 are displayed.

b. Now you can smile—it works!

Conclusion

We have seen how easy it is to configure RESTful Web services to send a JSON response to the client. The integration of JavaScript and Ajax made it easy to handle and parse the JSON message sent by the Web service. Finally, HTML5 provides very cool functionalities for storing the data in your browser and accessing the data with JavaScript after the page is loaded. This allows us to save time and bandwidth.

In Part 3 of this series, we will focus on the integration of Java API for XML Web Services (JAX-WS) technology for "big" Web services. `</article>`

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

```
@Path("/sellers/")
@Stateless
   public class SellersResource {
   @javax.ejb.EJB
   private SellerResource sellerResource;
   @Context
   ....
   @GET
   @Produces({"application/json"})
   public SellersConverter get(@QueryParam("start")
                               @DefaultValue("0")
...
```

See all listings as text

LEARN MORE

- Roy T. Fielding's dissertation defining the REST architectural style
- *HTML5: Up and Running*, by Mark Pilgrim (O'Reilly Media, 2010)

Agile ALM for Java Developers

Agile ALM makes for better-quality software and happier Java developers.

MICHAEL HÜTTERMANN

BIO

Agile application lifecycle management (ALM) is a way for Java developers, testers, release managers, technical project managers, and decision-makers to integrate flexible agile practices and lightweight tooling into software development phases.

Agile ALM synthesizes technical and functional elements to provide a comprehensive approach to common project activities and phases, addressing test, release, integration, build, requirements, change, and configuration management. (See **Figure 1**.)

With its interdisciplinary approach, agile ALM integrates project roles, project phases, and artifact types. Agile ALM enriches an ALM approach with agile values and strategies. An agile approach to ALM improves product quality, reduces time to market, and makes for happier developers.

In a nutshell, agile ALM

- Helps overcome process, technology, and functional barriers (such as roles and organizational units)
- Spans all artifact types as well as development phases and project roles
- Uses and integrates lightweight tools, enabling the team to collaborate efficiently without any silos
- Makes the relationship of given or generated artifacts visible, providing traceability and reproducibility
- Defines task-based activities that are aligned with requirements, which means the activities are linked to requirements and all changes are traceable to their requirements

As a result, from a Java developer perspective, agile ALM helps developers focus on doing their jobs rather than wasting time with barriers or obstacles that reduce productivity. Streamlining Java development with agile ALM fosters collaboration among coworkers, leading to improved software quality and better results.

Agile ALM provides processes and tool chains that are flexible and open to change. This is one of the ways in which ALM provides structure for agile software development. Some underlying aspects of agile ALM are not completely new, and developers should respect the struggles of previous decades, and put results together to get the best solution. In my view, ALM evolved from software configuration management (SCM), which in turn has its roots in basic version control.

Following are some major agile ALM building blocks that address and leverage widespread activities among Java developers.

Release Management

Release management involves producing software artifacts and releasing them according to a defined process. Release management can be differentiated into a functional part and a technical part. To deliver software successfully, both parts are important and should be integrated with each other. Automation and continuous integration are crucial facets of the software release and delivery process.

Functional release management. *Functional release management* involves identifying the customer's requirements, assigning them to releases, and delivering the functionality to the customer with high quality. Agile practices

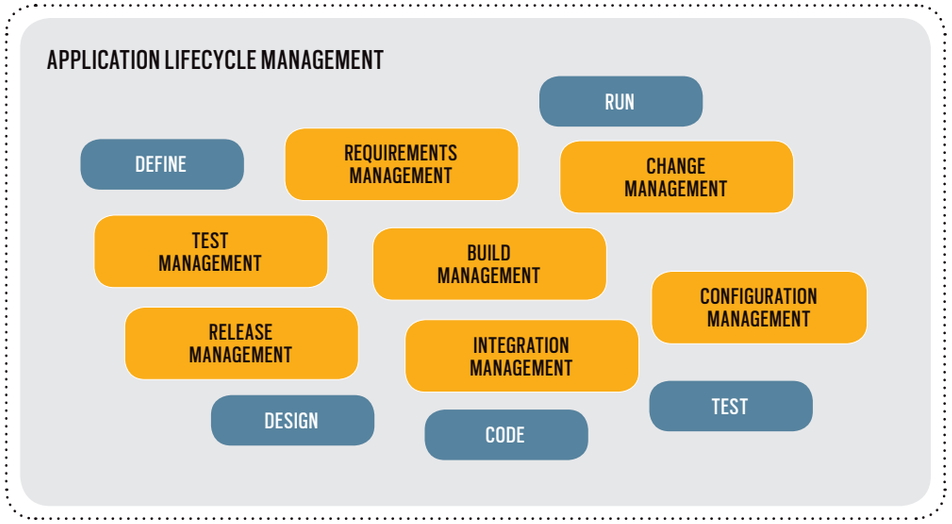


Figure 1

are often used to support this process, and many projects achieve good results from using the management template known as Scrum.

While defining a thin set of rules, Scrum fosters discipline and makes defects (in software as well as in the process) visible. But Scrum is too abstract to be applied "out of the book." You must implement and adopt it for software engineering. Implementation practices, for instance, might include distinguishing between special development phases, on a micro level, inside a Scrum release. Here are some examples:

- Opening the release with an interval to write acceptance tests
- Opening the release with an interval to get more-detailed requirements
- Closing the release with a frozen zone that allows developers to work only on bug fixes, not on new features
- Closing the release with a code-freeze interval in order to complete and ship the final release

Technical release management. Technical releasing consists of building the software and providing the final product to the customer. Build management (comprised of compiling scripts, packaging components, and distributing components) is essential for agile ALM.

Technical release management includes activities to identify configuration items, track and audit changes to requirements and configuration items, and integrate and deliver the implementation. In software engineering, change is more the rule than the exception. Because requirements change, it is important to keep the requirements in sync with their implementations. Possible gaps between functional and technical release management should be bridged. Strategies such as version control system (VCS) hooks help marry both parts of release management. A leading tool that supports technical release management is Maven. With Maven, the dependencies between your Java artifacts become visible.

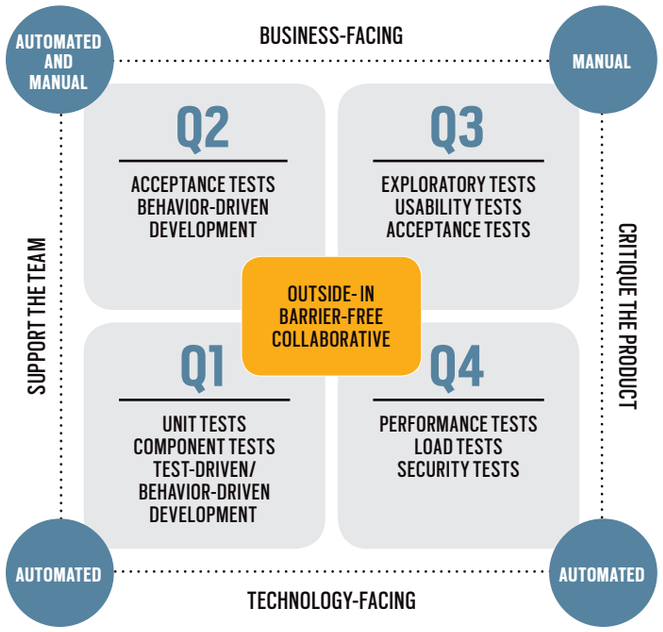


Figure 2

Automation and Continuous Integration

Automating manual steps involves delivering results in an objective and reproducible way. Automating the most error-prone, repetitive, and time-consuming activities is essential.

Continuous integration (CI) refers to the automation of the build, test, and release process with the goal of integrating the activities of colleagues and the work items others produce. This can result in a build ecosystem in which a new-code commit directly triggers a continuous build, including compilation, technical tests, audits, packaging, functional tests, and deployment.

FLEXIBLE STRUCTURE
Agile ALM provides **processes and tool chains** that are flexible and open to change.

All different artifact types, platforms, and languages (for example, Java, Groovy, Scala, PHP, COBOL, and others) should be integrated using a unified tool infrastructure. If no native build system exists for a respective language or platform, non-native build technologies can be used to include these artifact types in the CI farm.

One tool that can serve as your central tool hub is the build engine Hudson. Integrating Hudson with the tool Mylyn results in a tool chain that spans different project roles and activities. Then, the work on artifacts is traceable, starting with the source code in the developers' workspaces on up to tasks (work items) in your ticket system.

Using a component repository such as Nexus helps to differentiate between source code and Java binaries, and the repository operates on binaries separately—for example, through staging binaries from one test environment to another environment.

Collaborative Development

Writing software collaboratively means that all stakeholders, especially Java developers and testers, work on the solution constructively and with shared goals. The agile testing matrix (see **Figure 2**) defines test quadrants, which are integrated and aligned using the *outside-in* and *barrier-free* approach to software development.

The main drivers of the outside-in approach are

- Understanding your stakeholders and the business context
- Mapping project expectations onto outcomes more effectively
- Building more-consumable software, making systems easier to deploy and use
- Enhancing alignment with stakeholder goals continuously

Acceptance tests determine whether a system satisfies its specified acceptance criteria, which helps customers decide whether to accept the software. This means that acceptance tests also tell the programmers what the customer doesn't want them to do.

Executable specifications allow you to use tools that read the specifications automatically (either after manually starting the process or continuously as part of a CI process); process them against the system under test; and output the results in an objectively measurable, efficient, and readable way. The domain expert specifies the tests in simple

formats (for example, based on HTML or Microsoft Excel), and the program writes the results after running the tests against the system under test.

Describing requirements in an executable way fosters the barrier-free approach. Merging different documentation mediums (single-sourcing product information) reduces the amount of traditional documentation, because the specification is the system's functional documentation and, therefore, can be efficiently validated against the current software state. This leads to living documentation that is always up to date.

Behavior-Driven Development

Behavior-driven development (BDD) promotes a special approach to writing and applying acceptance tests that is different from the traditional *test-driven development* (TDD), although BDD also promotes writing tests first.

With BDD, tests are like functional specification stories, normally in a "given/when/then" format. This specification-oriented technique also uses a natural language to ensure cross-functional communication and to ensure that business concepts are understood. BDD provides a *ubiquitous language* for analysis and emphasizes application behavior over testing.

AGILE BENEFITS

Agile ALM improves product **quality**, reduces **time** to market, and makes for **happier developers**.

You can use dedicated, lightweight tools to write acceptance tests, or you can use specialized languages. Scala and Groovy, both first-class citizens on the Java Virtual Machine (JVM), offer interesting features for setting up a polyglot ecosystem and leveraging existing platforms by providing solutions that involve special-purpose languages. With Scala and Groovy, you can write tests, which helps to overcome

- Barriers between project phases and project activities (because coding and testing move together more closely)
- Barriers between artifact types (because code and executable specifications are written on the same unified infrastructure)
- Barriers between project roles (because tests are written collaboratively, with mechanisms to use terms close to the problem domain)
- Barriers between tools (because the same tools are used for programming and testing)

Overcoming common project barriers leads to what I call a *barrier-free* approach. The example shown in **Listing 1** gives you an impression of what it can look like to write BDD-flavored acceptance tests with Scala and the Scala specs2 library.

The code in **Listing 1** integrates the Web application test system, Selenium2, to smoke test the HotBot Web search.

The defined `is` method lists specification fragments that can be simple text (a description of the target system) or an example (including executable code that returns a result). Fragments are separated and linked by the ^ character.

LISTING 1

```
package alm

import org.specs2._
import matcher.ThrownExpectations
import specification.Step
import Thread._
import org.openqa.selenium.WebDriverBackedSelenium

class Selenium2 extends Specification with ThrownExpectations {

 lazy val selenium = new WebDriverBackedSelenium(new org.openqa.selenium.
firefox.FirefoxDriver(), "http://www.hotbot.com/")

 def is =
 sequential                                          ^
  "This specification smoke tests HotBot search"           ^
   Step(() => selenium)                                    ^
  "Enter search token and check the title of the SERP"      ! acc^
   Step(selenium.stop())                                   ^
                                      end
 def acc = {
  import selenium._
  open("/")
  `type`("search-Input", "Agile ALM")
  click("search-Submit")
  getTitle() must startWith("Agile ALM")
 }
}
```

See listing as text

Summary

Agile ALM provides more structure for agile software development and helps you approach ALM in a determined, pragmatic way. Using an agile approach to ALM can increase productivity and enhance results. Streamlining Java development with agile ALM integrates various practices (such as continuous integration and writing acceptance tests collaboratively), languages, and tools to improve software development practices. `</article>`

LEARN MORE

- *Agile ALM* (Manning Publications, 2011)
- *A Guide to Software Configuration Management* (Artech House, 2000)
- *Configuration Management Principles and Practice* (Addison-Wesley, 2003)

WINSTON PRAKASH AND
SUSAN DUNCAN

BIO

Understanding the Hudson Plug-in Development Framework: Part I

Use the Hudson Plug-in Interface tool to create, build, run, and debug custom plug-ins that extend Hudson to meet the specific needs of your projects.

This article is for beginners who are interested in understanding the fundamentals of Hudson plug-in development. Hudson is a popular open source continuous integration (CI) tool written purely in Java. Apart from being an open source product, the popularity of Hudson is due to its extensible nature, which uses plug-ins and the plug-in developer ecosystem.

Plug-ins allow developers to do everything from customize the way builds are done and results are displayed to enable integration with application lifecycle management (ALM) systems such as software configuration management (SCM), testing and analysis tools, and so on. More than 400 Hudson plug-ins, supporting various aspects of CI, are available for free installation.

Hudson provides a series of extension points that allow developers to extend Hudson's functionality. The Hudson Plug-in Interface (HPI) tool helps developers create, build, run, and debug plug-ins.

This article is Part 1 of a two-part series and covers the following topics:

- Creating a Hudson plug-in using the HPI tool
- Extending a Hudson extension point to provide a custom implementation

Part 2 will cover the following topics:

- Writing a configuration for a Hudson extension
- Adding a global configuration for a Hudson extension

POPULAR TOOL
Hudson's popularity is due to its **extensible nature**.

Creating and Tuning a Hudson Plug-in

First, we'll look at how to create a Hudson plug-in using the HPI tool, create and run a project for the plug-in, and view the plug-in in action on the Hudson test server.

Tip: The HPI tool is a Maven plug-in. Hudson extensively uses Maven, another popular open source software project management and comprehension tool, which can be downloaded from maven.apache.org.

Generating the skeleton plug-in. The first step is to create the skeleton plug-in, which is a bare-bones plug-in. The following simple command tells Maven to create the Hudson plug-in skeleton's source code using the HPI tool:

```
mvn hpi:create
```

Maven downloads all the required software to execute the command and prompts you for the groupID and artifactID.

```
Enter the groupId of your plug-in:
org.sample.hudson
Enter the artifactId of your
plug-in:
sample-plug-in
```

Tip: Maven might throw the following error:

```
[ERROR] No plug-in found for
prefix 'hpi' in the current
project ..
```

If it does, add the following entry in the ~/.m2/settings.xml file:

```
<plug-inGroups>
 <plug-inGroup>org.jvnet.hudson.
 tools</plug-inGroup>
</plug-inGroups>
```

The name of the generated folder depends on the value you entered for the artifactId. The file structure of the generated folder will have the following layout:

- pom.xml: The Maven Project Object Model (POM) file, which is used to build the plug-in
- src/main/java: The folder that contains the Java source files for the plug-in
- src/main/resources: The folder that contains the Jelly UI

PHOTOGRAPH BY BOB ADLER

framework's view files for the plug-in
- src/main/webapp: The folder that contains static resources for the plug-in, such as images and HTML files

Building and running a plug-in project. The next step is to build a project for the plug-in by running the following command:

```
mvn package
```

The project that is created is minimal, but it is a complete Maven project. It can be run without any modification using Maven.

The package command tells Maven to build the project and create an HPI package that can be installed directly to a Hudson server.

The skeleton plug-in project has a sample extension, which is fully functional. You can run the project and see the results of the implementation provided by the plug-in developer for an extension point by running the following command:

```
mvn hpi:run
```

Because the plug-in project is a Maven project, the configuration of the project is defined in the pom.xml file. The Maven goal hpi:run is responsible for performing several tasks, including starting the Jetty server, adding Hudson as a Web application to that server, and installing the plug-in to Hudson.

The work subfolder under the plug-in project folder is used as Hudson Home. The work/plug-ins folder contains a list of .hpi files corresponding to various bundled plug-ins. The only notable exception is the .hpl file, which corresponds to the plug-in project being run.

The .hpl file is a simple text file that contains metadata describing all the items (classes, JAR files, and resources) associated with the currently built Hudson plug-in. This file is generated by the HPI tool every time the plug-in project is run by the hpi:run command. Hudson knows how to interpret this file and load the entire plug-in without packaging the plug-in as a .hpi package, which makes it easy to do debugging during development.

Winston Prakash and Susan Duncan discuss Hudson while onsite at JavaOne 2011.

Once the plug-in project has been run successfully and the Jetty server has been started, you can view the Hudson main page by pointing a browser at the following address:

```
http://localhost:8080
```

Examining the sample extension. As mentioned previously, the hpi:run command installs the plug-in to Hudson and adds the plug-in to the Jetty server as a Web application. This plug-in does the following tasks:
- Adds an extension to the Hudson Builder interface. This sample custom builder is called HelloBuilder, and it does not do anything fancy. It simply prints the text Hello <name> to the build console log.
- Provides a UI for configuring the HelloBuilder extension.

Figure 1

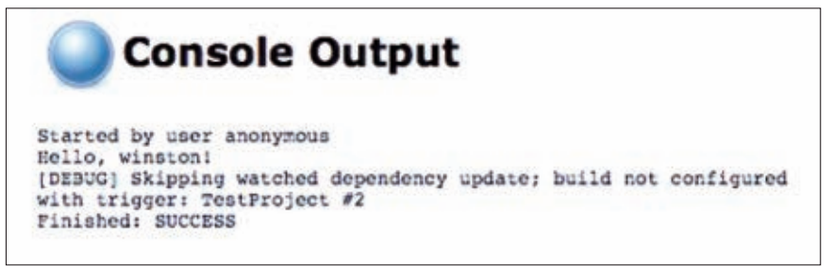


Figure 2

It's easy to see HelloBuilder in action by creating a simple Hudson project—for example, TestProject—and then configuring HelloBuilder.

You can add HelloBuilder as a builder to TestProject by using a drop-down menu in the Build section of the configuration page. After you add HelloBuilder as a builder for the project, the Build section shows HelloBuilder as one of the builders.

The task of HelloBuilder is to print the message Hello <name> to the console log, so enter a name in the provided **Name** text box.

Because HelloBuilder is set as the only builder of the TestProject project, when you start a build, HelloBuilder is asked to perform its task. Once the build is completed, you can view the result of HelloBuilder in the build console output.

Extending the Builder Extension Point

Now let's look at some of the code that extends the builder extension point so you understand how to do the following:

- Extend an extension point.
- Implement the methods to extend the functionality encapsulated by an extension point.

Hudson provides the concept of *extension points* and *extensions* to facilitate using plug-ins to add functionality to the core platform. Extension points are interfaces that encapsulate entry points to extend the functionality of a service provided by the core platform.

Among the various services provided by Hudson, the foremost is building a job. A job is a buildable project that consists of several configurable areas and build steps. The following are some of the build steps:

- SCM checkout: Based on the SCM type, the source code is checked out.
- Prebuild: This step is invoked to indicate that the build is starting.
- Build wrapper: This step prepares an environment for the build.
- Builder runs: This step causes the actual building to occur, much like calling Ant or Make.
- Recording: This step records the output from the build, such as test results.
- Notification: This step sends out notifications that are based on the results determined so far.

Builders are responsible for building jobs. The extension point provided by Hudson for contributing to the Builder runs step is aptly called Builder.

Hudson comes bundled with two of the most-popular builders: Ant and Maven. They are in fact extensions to the Builder extension point. So it is possible for a plug-in to provide its own builder extension as one of the builders of the job.

Several external plug-ins exist for other popular builders, such as Make, Gradle, Rake, and so on. HelloBuilder, our example builder extension, is a contrived example to demonstrate how extensions are built. Far more-sophisticated builder extensions are possible using the Builder extension point. Let's examine the source code to understand how the extension mechanism works.

FAST FACT

Hudson provides the concept of **extension points and extensions** to facilitate using plug-ins to add functionality to the core platform.

Tip: Even though in this article the sample extension is referred to as HelloBuilder, the Java class corresponding to it is called HelloWorldBuilder.java.

Examining the HelloBuilder extension. In order for Hudson to understand a class as an extension, you must do the following:

- Extend a class that advertises itself as an extension point
- Implement the required abstract methods to extend the functionality
- Tell Hudson that the particular class is an extension

Looking at the source code for HelloWorldBuilder.java, you can see that the class HelloWorldBuilder extends the class Builder, which is the extension point for the Hudson Builder interface.

LISTING 1 LISTING 2 / LISTING 3

```
public boolean perform(AbstractBuild<?, ?> ab, Launcher launcher,
BuildListener bl) throws InterruptedException, IOException;
```

See all listings as text

```
public class HelloWorldBuilder
extends Builder {
```

The Builder class itself is a subclass of BuildStep, which defines the abstract method that needs to be implemented by the extension to contribute to the Builder interface. The abstract method that needs to be implemented by any builder extension is shown in **Listing 1**.

BuildStep.perform(..) overridden by HelloBuilder will be called by Hudson to include the BuildStep functionality extended by the HelloBuilder extension.

Finally, to tell Hudson the class is an extension to some extension point, you must annotate the class with the @Extension annotation. In **Listing 2**, the @Extension annotation at the inner class DescriptorImpl tells Hudson the class is an extension.

Examining the BuildStep.perform(..) abstract method. The BuildStep .perform(..) abstract method provides access to three objects:

- Build: An object representing the build of the job being performed. Build in turn provides access to important model objects, such as the following:
  - Project: The buildable job
  - Workspace: The folder where the build happens
  - Result: The result of the build up to this build step
- Launcher: An object that is used to launch the build of this job
- BuildListener: An interface for communicating the status of the build steps being performed by this builder and for sending any console messages from the build steps to Hudson

HelloBuilder uses the BuildListener model object to print the Hello message to the console, as shown in **Listing 3**.

listener.getLogger() gets the logger from the BuildListener object whose output goes to the console. The code simply prints Hello <name> via the logger.

Modifying the HelloBuilder perform(..) method. Because the BuildStep .perform(..) method provides access to

the Launcher object, it is easy to

- Use the Launcher object to execute an external executable.
- Send the results of the execution to the console.

These tasks are done by adding the code shown in **Listing 4** to the BuildStep .perform(..) method.

Running the plug-in project again shows the console output shown in **Listing 5**.

If there is an error, the exception corresponding to the error is displayed, as shown in **Listing 6**.

Analyzing the HelloBuilder perform(..) method. The code added to perform(..) is contrived, but it explains some of the important concepts. For example:

- When a build step is started or stopped, let Hudson know about it. This is done via the Job Build Listener interface.

```
listener.started(buildStepCause);
..
..
listener.finished(Result.SUCCESS);
```

It is important to inform Hudson for two reasons:

- Hudson heuristically shows the progress of the overall build of the job.
- When a build step fails, Hudson must stop the overall progress of the build and mark the build as FAILED. This is done by sending a message to Hudson about the status of the build via BuildListener.

- Use the Launcher interface to launch your external executable. Send the console output of your execution to Hudson, as shown in **Listing 7**.

Launcher correctly launches the application in the master or slave node the job is running.

Always use the return status of the Launcher to find out whether the execution was successful. The standard output of the Launcher is hooked to the listener. This sends console output from the execution to Hudson, and this is how the output of the command to list the user directory is displayed in the build console.

- Notify Hudson of any failure of the Build step, as shown in **Listing 8**. The StackTrace of the exception is sent to Hudson via Exception .printStackTrace(listener.fatalError(..)).

Summary

The Hudson Continuous Integration Server is a popular open source project that recently became a technology project at Eclipse Foundation. It is supported by an ecosystem of plug-in developers who develop plug-ins for various aspects of the Hudson CI system.

The Hudson plug-in development environment provides a rich set of extension points with which plug-in developers can develop custom plug-ins. This article explored the fundamentals of the plug-in development environment by explaining the various steps involved in developing a simple plug-in using Hudson's HPI tool.

Part 2 of this series will explore ways to configure the extensions in a plug-in, describe how to use the Jelly UI framework for configuration, and show how to create a global configuration for an extension. `</article>`

LISTING 4 | LISTING 5 | LISTING 6 | LISTING 7 | LISTING 8

```
List<Cause> buildStepCause = new ArrayList();
buildStepCause.add(new Cause() {
   public String getShortDescription() {
          return "Build Step started by Hello Builder";
   }
  });
listener.started(buildStepCause);
ArgumentListBuilder args = new ArgumentListBuilder();
if (launcher.isUnix()) {
   args.add("/bin/ls");
   args.add("-la");
} else {
     args.add("dir"); //Windows
}

String homeDir = System.getProperty("user.home");
args.add(homeDir);

try {
   int r;
   r = launcher.launch().cmds(args).stdout(listener).join();
   if (r != 0) {
      listener.finished(Result.FAILURE);
      return false;
   }
} catch (IOException ioe) {
   ioe.printStackTrace(listener.fatalError("Execution" + args + "failed"));
   listener.finished(Result.FAILURE);
   return false;
} catch (InterruptedException ie) {
   ie.printStackTrace(listener.fatalError("Execution" + args + "failed"));
   listener.finished(Result.FAILURE);
   return false;
}
listener.finished(Result.SUCCESS);
```

See all listings as text

LEARN MORE

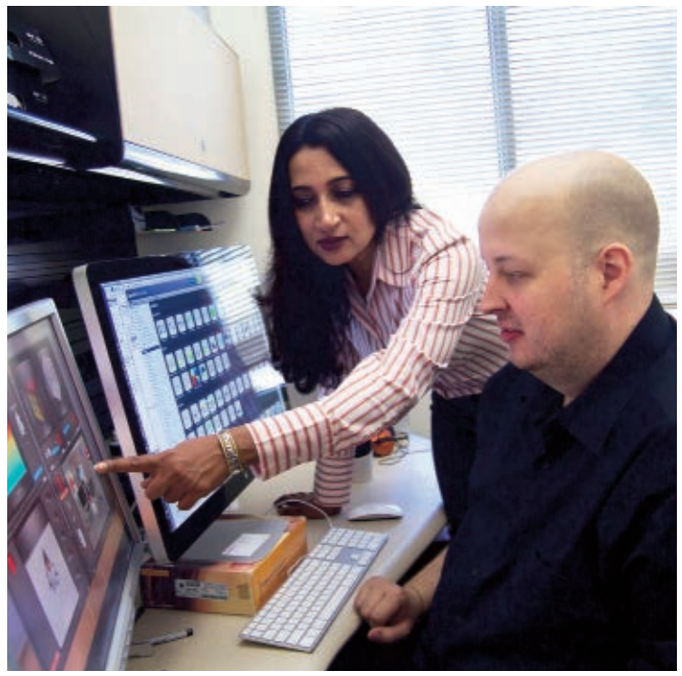
- Hudson Website
- Maven Website

JavaFX 2.0

Shock the Senses

Oracle's **Nandini Ramani** talks with *Java Magazine* about the key features of JavaFX 2.0. **BY MICHAEL MELOAN**

JavaFX 2.0 is Oracle's premier development environment for rich client applications. Designed to leverage Java's APIs and cross-platform flexibility, JavaFX 2.0 provides state-of-the-art solutions for creating optimal UIs. Nandini Ramani, vice president of Java client development at Oracle, discusses how this new Java technology will help developers to build advanced systems.



PHOTOGRAPHY BY BOB ADLER
ART BY PAULINA MCFARLAND

Java Magazine: What are the goals for JavaFX 2.0? And what advantages does it offer system architects and developers?

Ramani: JavaFX 2.0 is a major modernization for client-side developers. At the top of the stack, we have more than 6,500 new APIs for building graphically rich, compelling user interfaces. These APIs

Nandini Ramani (center) and members of the JavaFX 2.0 team discuss new capabilities.

RICH HISTORY
JavaFX 2.0 reflects **years of rigorous** development, testing, and cross-platform capability.

include everything from the ability to play audio and video, to user interface controls and layout containers, to a scene graph and animation libraries, to observable collections and UI binding. At the bottom of the stack, we have a new windowing library and hardware accelerated graphics library.

The industry is moving toward multicore multithreading platforms with GPUs [graphics processing units], and JavaFX 2.0 leverages these attributes to improve execution efficiency and UI design flexibility. Our initial goal is to give architects and developers of enterprise applications a set of tools to help them build better business applications, dashboards, richly animated UIs, and mashups that integrate different Web-based datasources.

Java Magazine: Software developed using JavaFX 2.0 can be run standalone, in the browser, or using Java Web Start. Will you describe application areas best suited to each of these options?

Ramani: The standalone mode is appropriate when higher control on the runtime environment is appropriate. The application is launched from a client-side JAR file, or perhaps bundled as a traditional OS-specific executable by using existing third-party tools. When executing in the browser, the JavaFX application is embedded in a Web page using the traditional plug-in model. With Java Web Start, the application is run on the desktop, but it's initially downloaded from the Web, and like a Web application, it can be updated every time the application is run. This simplifies deployment and update logistics by downloading from a central Web location, and it handles updates automatically.

Java Magazine: What strengths does JavaFX 2.0 offer over Adobe Flash or Microsoft Silverlight?

Ramani: First and foremost, JavaFX 2.0 is built on top of the Java platform, so it reflects all the years of rigorous development, testing, and cross-platform capability that Java brings to the table. It's hosted on the efficient and widely available Java Virtual Machine. Many users of JavaFX 2.0 will already be using Java EE for back-end processing, so JavaFX on the front end allows them to utilize existing skill sets, tooling, and infrastructure. Everything they apply to back-end development becomes applicable on the front end. In addition, Java has the largest and strongest

Watch *Introducing JavaFX 2.0*

ecosystem in terms of third-party libraries around. Being able to leverage that kind of support for developing client applications is a big plus.

Java Magazine: Can languages like JRuby, Scala, and Groovy interoperate with JavaFX 2.0?

Ramani: Absolutely. Just as you can run any of these languages on top of the Java platform, they also all interoperate with JavaFX 2.0. In fact, there's a project in the Groovy community called GroovyFX, which uses the Builder design pattern for JavaFX applications. Most of this work is being done open source, and many of the language implementers are excited about using JavaFX 2.0 technology. For some Java developers, dynamic scripting languages are a vital part of their toolkit, and JavaFX can seamlessly work with them.

Because JavaFX uses familiar design patterns, such as POJOs following the JavaBeans naming conventions, many of these languages already have enhanced support for JavaFX right out of the gate. In addition, JavaFX was carefully designed with consistent API idioms, such as event handling, which make it easy for other languages to provide syntactic sugar for these APIs.

Java Magazine: Will you describe an application scenario where Prism, the hardware accelerated graphics pipeline, and the new Glass windowing toolkit might offer tangible advantages?

Ramani: Prism handles rasterization and rendering of JavaFX scenes. It can execute on both hardware and software renderers. The Glass windowing toolkit is the lowest-level framework for the JavaFX 2.0 graphics stack—it manages timers, surfaces, and windows.

When developers need gradients and shading, or any sort of advanced graphics, hardware acceleration can offer a huge payoff in system responsiveness. And when applications need to build tables and display large data sets, Prism can also be valuable. The Prism/Glass combination eliminates the need to instantiate and load Swing and AWT classes, which gives JavaFX 2.0 a definite performance advantage.

Java Magazine: How will FXML, the JavaFX 2.0 XML-based markup language, add value for developers? And will you describe how Scene Builder will be useful for generating FXML?

Ramani: The fact that FXML is XML-based is a huge advantage from a tools perspective, because Java IDEs such as NetBeans and Eclipse interoperate easily with XML. Layout design for enterprise applications is one of the trickiest and most-labor-intensive endeavors. The visual JavaFX Scene Builder tool mitigates the need for hand coding, which is a definite productivity bonus. Also, FXML ties into JavaFX Scene Builder in two ways: developers can either programmatically make changes and then generate FXML, or they can use JavaFX Scene Builder to design the JavaFX UI components and then save them as FXML markup. JavaScript can also be seamlessly integrated into this scenario. So there's a great deal of flexibility. These tools also allow the use of CSS. Once an application is created, a completely new visual look can be generated without recompilation by changing either the style sheet or the FXML code. This is an industry trend, and JavaFX 2.0 provides the toolset.

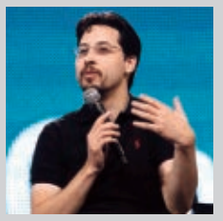
Java Magazine: Is HTML5 part of the JavaFX 2.0 landscape?

Ramani: HTML5 technologies are still under development by the W3C. But support for HTML5, as defined today, is built into JavaFX 2.0. Our Web component is based on WebKit, which supports HTML5. Many developers are already using WebKit-based browsers. Possible future developments, like offline storage, Web SQL databases, native drag-and-drop, and geolocation, will be of interest as they evolve. We plan to stay in sync with HTML5.

Java Magazine: How do the JavaFX UI control libraries fit into the JavaFX 2.0 toolset?

Ramani: I was a longtime Swing developer myself, and I love its capabilities—a great way to program. But every component in Swing tends toward rectangular forms. Today, we need more flexibility. JavaFX 2.0 UI controls offer the ability to create rounded shapes, a much wider range of components, and

COMMUNITY COMMENT



"Making JavaFX open source is a huge step in the right direction for this technology. It will enable businesses to use JavaFX with no fear about vendor lockin, and let the community participate and grow the platform in new and innovative ways. I have been advocating for this move since the 1.0 release, and am glad that Oracle listened to the community and made this possible."

—Stephen Chin, a technical expert in RIA technologies and chief agile methodologist at GXS

PHOTOGRAPH BY HARTMANN STUDIOS

Nandini Ramani and members of the JavaFX 2.0 team discuss the product's roadmap.

LEAPS AND BOUNDS

JavaFX 2.0 **is a leap forward**. Developers can seamlessly **mix and match** Web and media content within their Java applications.

skinning via CSS. This new set of powerful components will allow any developer to build advanced enterprise application interfaces. Also, the JavaFX UI controls will be the first component of JavaFX to be open sourced as an OpenJDK project. This will enable developers to build their own custom versions of components, which the Swing community has been doing for many years.

Java Magazine: Can JavaFX 2.0 code be easily integrated into existing Swing applications?

Ramani: Yes, Swing and JavaFX cooperate well. Within existing Swing applications, you can either add on or embed JavaFX. All Swing capabilities are retained, but the portions rendered with JavaFX leverage all the new capabilities. For instance, if you want to embed a Web browser into your Swing application, you can do so by embedding the JavaFX WebView. Another option is to rewrite some of the Swing code for JavaFX. It's a great migration path for existing Swing applications.

Java Magazine: What operating systems will be supported by JavaFX 2.0?

Ramani: The first release will support 32-bit and 64-bit versions of Microsoft Windows XP, Windows Vista, and Windows 7. In addition, we'll be releasing a developer preview of the JavaFX 2.0 SDK for Mac OS X. A Linux version will be made available at a later date.

Java Magazine: How will JavaFX evolve going forward?

Ramani: On the client side, we're trying to stay in lockstep with the evolution of the Java platform. We will integrate new features as they emerge, like invokedynamic and NIO.2. Two of the biggest enhancements to the Java platform planned for Java SE 8 are modularization and lambda expressions. JavaFX APIs were specifically designed with both of these features in mind. For example, the event handling APIs are all designed around SAM [single abstract method] interfaces, which happen to be the exact type of interface that the lambda proposal for Java SE 8 requires.

But we don't have to look to the future for something exciting. JavaFX 2.0 is a leap forward. It offers developers the ability to seamlessly mix and match Web and media content within their Java applications using a variety of powerful coding and layout options, which can be running inside a browser or as standalone applications. We believe that a broad range of enterprise applications will benefit from this multifaceted new technology.

Java Magazine: What role will open source play in JavaFX's future?

Ramani: At JavaOne, we announced our intention to submit a proposal to open source the JavaFX platform as a new OpenJDK project. We plan to initially contribute the JavaFX UI controls and related libraries; other JavaFX components will follow in multiple phases. By the time you read this article, we are confident that the OpenJDK community will have approved this proposal. You can check the status of the JavaFX project on the OpenJDK site. `</article>`

Michael Meloan began his professional career writing IBM mainframe and DEC PDP–11 assembly languages. He went on to code in PL/I, APL, C, and Java. In addition, his fiction has appeared in *WIRED*, *BUZZ*, *Chic*, *LA Weekly*, and on National Public Radio. He is also a Huffington Post blogger.

LEARN MORE

- JavaFX home page
- JavaFX 2.0 downloads
- FX Experience
- Overview of JavaFX

SIMON RITTER

BIO

JavaFX and Swing Integration

Use the power of JavaFX 2.0 to migrate Swing interfaces to JavaFX.

JavaFX 2.0 made its debut at JavaOne in October 2011 (see "JavaFX 2.0 Is Here!" page 3). JavaFX 2.0 is an advanced Java UI platform for business and enterprise applications, and it represents the next step in the evolution of Java as a premier rich client platform.

JavaFX 2.0 reduces the learning curve for experienced Java programmers because all that's required is learning a new set of APIs and gaining a grasp of use binding, timeline-based animations, a scene graph, and so on. In addition, JavaFX 2.0 reverses the approach to Swing and JavaFX integration.

This is the first in a series of articles that will show how to modify an existing Swing-based application so that the interface can be migrated from Swing to JavaFX. This series will show how parts of the application can be replaced gradually with JavaFX to improve the appearance and usability of the application.

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

WHAT'S NEXT

JavaFX 2.0 represents the **next step in the evolution** of Java as a premier rich client platform.

Important Things to Know About JavaFX

Because JavaFX 2.0 is a set of Java APIs, using it for a project within an IDE is simply a matter of including the JavaFX Runtime file (jfxrt.jar) in the classpath.

Having done that, you can go about developing the components you want to use in your combined Swing and JavaFX application.

Note: For this series of articles, we won't delve into the mechanics of JavaFX. See the Learn More section for JavaFX resources.

Sample Swing Application

To demonstrate the power of JavaFX, we'll use a relatively simple Swing application that has enough complexity in its components but can be easily understood and modified. The first application is called StocksMonitor, and it was created by Hirondelle Systems. It monitors stock market trades; displays the data in a table; and allows users to create, edit, and delete portfolios of stocks. StocksMonitor is available through javapractices.com and is distributed under a creative commons license.

Integration Basics

First, let's look at what's required to treat a JavaFX scene as a Swing component. Swing uses a container-component hierarchy with layout managers to organize the way individual components are displayed on the screen. JavaFX uses a scene graph, but as long as we can make our JavaFX scene appear to be a Swing component, the differences are of no direct consequence.

Thankfully, most of the hard work is taken care of by the JavaFX platform. Included in the JavaFX API is the JFXPanel class. The name is a little misleading, because this class extends the Swing JComponent class rather than the Abstract Windows Toolkit (AWT) Panel class. As such, we can use this easily in a Swing application, but we can't use it in a pure AWT application.

Jasper Potts, the developer experience architect for Java client, talks about the JavaFX 2.0 release.

PHOTOGRAPH BY BOB ADLER

Listing 1 is a simple example of using this class. Let's work through the details, because some things aren't obvious.

In line 8, we instantiate a JFXPanel object, but initially we tell it nothing about what it will contain. That information will be provided later. This component can be added to a Swing Container just as any other Swing component.

To understand how we encapsulate the JavaFX scene, we need some details about threads in Swing and JavaFX. Swing uses a single event dispatch thread (EDT), and JavaFX uses an application thread. Both effectively do the same thing, which is to process asynchronous events that affect the UI (for example, button presses, updates to displayed values, window resizing, and so on).

As a Swing component, the JFXPanel object must be accessed only from the Swing EDT. The only exception to this is the setScene() method, which must be accessed from the JavaFX application thread.

In order to set the scene for the JFXPanel object, we must create another object that implements the Runnable interface, which can then be posted to the event queue and executed at some (unspecified) time in the future. We use the static utility method, runLater(), provided in the Platform utility class, and then an anonymous inner class to instantiate the Runnable object. Due to the inner class restrictions, the reference to the JFXPanel must be made final.

The initFXComponent() method is where the real JavaFX work happens. Because JavaFX uses a scene graph, we need to create a Scene object to hold the parent node reference. When we instantiate this object, we can also set the size of the scene in pixels, which will then be used by the Swing layout manager to position the JavaFX component. We also set the root node of the scene to be the graph of objects that will be displayed.

If we wish to include more than one JavaFX scene in our Swing application, we simply instantiate more JFXPanel objects. To optimize the use of multiple JavaFX scenes, use a single call to Platform.runLater() and have the run() method of the inner class perform initialization for all scenes.

Swing and JavaFX Events

There is one more basic part of Swing and JavaFX integration that needs to be addressed, which is how the components interact when a Swing component triggers a change in a JavaFX scene or vice versa. If the JavaFX scene wants to interact with Swing components, the code must be wrapped in a Runnable object and placed on the EDT using the SwingUtilities.invokeLater() method, as shown in **Listing 2**.

For the opposite case, where a Swing component needs to interact with a JavaFX scene, the code must again be wrapped in a Runnable object, but this time it must be placed on the JavaFX application thread using the Platform.runLater() method, as shown in **Listing 3**.

Conclusion

So far, we've seen that adding a JavaFX scene to a Swing container requires only a few more lines of code than using a normal Swing component. Having the JavaFX scene and Swing components interact is also straightforward. You just have to remember to wrap the necessary code in a Runnable object and place it on the correct queue for execution.

In the next article, we'll take this knowledge and start to modify our Swing application to add exciting new components that are created using JavaFX. `</article>`

LISTING 1 LISTING 2 / LISTING 3

```
1 public void init() {
2   setSize(300, 200);
3   setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
4   setLayout(new BorderLayout());
5
6   /* Put the JavaFX scene in the middle */
7   fxComponent = new SimpleFXComponent(this);
8   final JFXPanel fxPanel = new JFXPanel();
9   add("Center", fxPanel);
10
11  /* Construct the JavaFX scene on its own application thread */
12  Platform.runLater(new Runnable() {
13    @Override
14    public void run() {
15      Scene scene = new Scene(fxComponent, 200, 100);
16      scene.setFill(Color.BLUEVIOLET);
17      fxPanel.setScene(scene);
18    }
19  });
20
21  /* Put the Swing button at the bottom */
22  changeButton = new JButton("Change JavaFX Button");
23
24  ...
25
26  add("South", changeButton);
27  setVisible(true);
28 }
```

See all listings as text

LEARN MORE

- Documentation and tutorials
- API documentation (Javadoc)
- JavaFX Showcase

Using Transitions for Animation in JavaFX 2.0

Animate the nodes in your scene the easy way.

JAMES L. WEAVER

BIO

JavaFX 2.0 is an API and runtime for creating rich internet applications (RIAs). JavaFX was introduced in 2007, and version 2.0 was released in October 2011. One of the advantages of JavaFX 2.0 is that the code may be written in the Java language, using mature and familiar tools. This article focuses on using JavaFX 2.0 *transitions* to animate visual nodes in the UI.

JavaFX comes with its own transition classes, shown in **Figure 1**, whose purpose is to provide convenient ways to do commonly used animation tasks. These classes are located in the javafx.animation package. This article contains an example of using the TranslateTransition class to animate a node, moving it back and forth between two positions in the UI.

Overview

To help you learn how to use the TranslateTransition class, an example named TransitionExampleX will be employed. As shown in **Figure 2**, this example contains a couple of buttons, a couple of rectangles, and a circle that you'll animate a little later.

Class	Description
TranslateTransition	Translates (moves) a node from one location to another
RotateTransition	Rotates a node
ScaleTransition	Scales (increases or decreases the size of) a node
FadeTransition	Fades (increases or decreases the opacity of) a node
PathTransition	Moves a node along a geometric path
SequentialTransition	Allows you to define a sequential series of transitions
PauseTransition	Used in a SequentialTransition to wait for a period of time
ParallelTransition	Allows you to define a parallel series of transitions

Figure 1

The TransitionExample project that you'll download in the next section contains starter code for this example, and it has an appearance at runtime similar to **Figure 2**. During the course of this article, you'll modify the code to implement the animated behavior of the TransitionExampleX project, which is also available in the download.

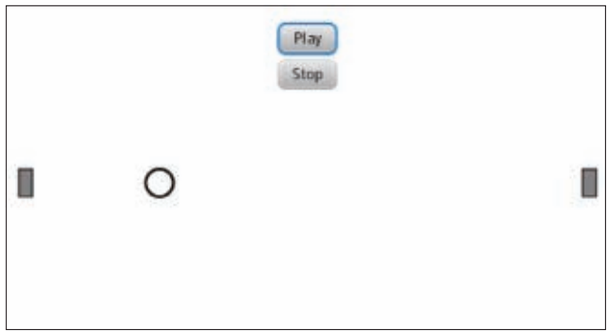


Figure 2

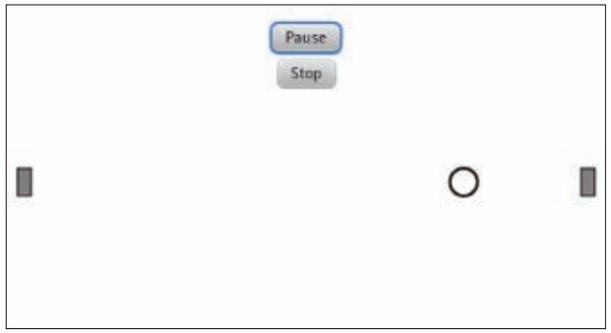


Figure 3

When you click the Play button, the text on the button changes to "Pause," as shown in **Figure 3**, and the ball moves back and forth between the paddles indefinitely.

Clicking the Pause button causes the animation to pause and the text on the button to change to "Play."

Obtaining and Running the TransitionExample Example

1. Download the NetBeans project that includes the TransitionExample program.
2. Expand the project into a directory of your choice.
3. Start NetBeans, and select **File -> Open Project**.

PHOTOGRAPH BY STEVE GRUBMAN

This is an update to an article that was originally published on Oracle Technology Network (July 2011).

4. From the Open Project dialog box, navigate to your chosen directory and open the TransitionExample project, as shown in **Figure 4**. If you receive a message dialog 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.

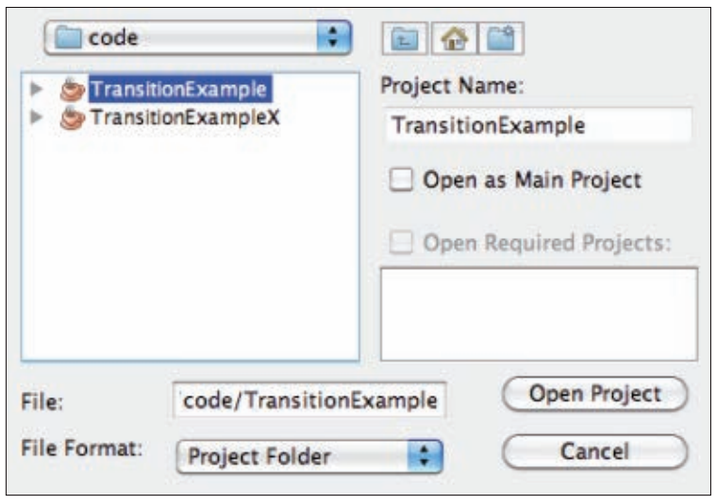


Figure 4

Figure 5

Figure 6

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

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

You'll notice that clicking the Play and Stop buttons has no effect. Your mission will be to add code that implements the behavior described previously. Here are steps you can follow to implement this behavior:

Step 1: Create a TranslateTransition Instance to Animate the Node

To cause the ball to move (also known as *translate*) between two different positions in the scene, we'll create an instance of the TranslateTransition class. Take a look at the code in the TransitionExampleMain.java file in the TransitionExample project, which shows the starter code for this example.

Using the TranslateTransitionBuilder class to build TranslateTransition. The starter code in TransitionExampleMain.java makes use of *builder* classes in the JavaFX 2.0 API, including the TranslateTransitionBuilder class shown in **Listing 1**.

LISTING 1 | LISTING 2 | LISTING 3 | LISTING 4

```
transition = TranslateTransitionBuilder.create()
 // TO DO: Insert code to set the duration to 1500 milliseconds

 .node(ball)
 // TO DO: Insert code to set the fromX property to 0

 // TO DO: Insert code to set the toX property to 440

 .interpolator(Interpolator.LINEAR)
 // TO DO: Insert code to set the autoReverse property to true

 .cycleCount(Animation.INDEFINITE)
 .build();
```

See all listings as text

Go ahead and fill in the lines indicated by the "TO DO" comments, so the code in **Listing 1** turns into the code shown in **Listing 2**.

There are many builder classes in the JavaFX API, and their purpose is to enable a declarative style of programming to create and set the properties of objects. For example, the code you completed in **Listing 2** creates an instance of the TranslateTransition class, and it populates that instance with properties such as the duration of the animation and the node in the scene graph being animated. As you can see in TransitionExampleMain.java, other builder classes used in this application include ButtonBuilder, CircleBuilder, RectangleBuilder, SceneBuilder, and VBoxBuilder.

Step 2: Define an Event Handler in the Play Button

To make the TranslateTransition start, pause, and stop, you'll define event handlers in the buttons that call the appropriate methods on the TranslateTransition object. For example, the starter code in **Listing 3** from TransitionExampleMain.java contains the builder that creates a Button instance for the Play button.

As you did before, fill in the lines indicated by the "TO DO" comments, turning the code shown in **Listing 3** into the code shown in **Listing 4**.

Using methods of the Animation class to control the animation. All the transition classes in **Figure 1** are subclasses of the

Transition class, which is a subclass of the Animation class. The Animation class has methods, such as start() and getStatus(), that control and monitor the state of the animation. As a result of the code that you completed in **Listing 4**, when the button is clicked, the status of the animation is ascertained. If the animation status is PAUSED, the play() method of the animation is invoked; otherwise, the pause() method is called.

JAVA POWER

One of the **advantages of JavaFX 2.0** is that the code may be written in the Java language, using mature and familiar tools.

Using binding to keep properties updated. Property binding is a very convenient and powerful feature of the JavaFX API, because it enables you to keep the values of properties automatically updated. The excerpts from **Listing 1** that are shown in **Listing 5** bind the textProperty of the playButton to a StringProperty that holds the text to be displayed in the button.

As the value of playButtonText is updated by the event handler that you coded, the text on the button displays the updated value.

There is another use of property binding in TransitionExampleMain.java, which is shown in **Listing 6** and demonstrates how to bind expressions.

To keep the buttonsContainer node horizontally centered in the application window, the layoutXProperty of one of the nodes in the scene graph is bound to an expression that contains the widthProperty of the Scene.

Note: The excerpt in **Listing 6** is for demonstration purposes only, because the use of layout containers (in the javafx.scene.layout package) is the preferred approach for dynamically positioning nodes within a scene.

Step 3: Define an Event Handler in the Stop Button

To finish the TransitionExample, you'll need to enter some code into the starter code from TransitionExampleMain.java shown in **Listing 7**.

The completed code for the Stop button event handler is very similar to what you've already coded in the Play button event handler. Go ahead and fill in the code shown in **Listing 8**, and run the example.

Conclusion

JavaFX 2.0 comes with several transition classes that extend the Transition class, whose purpose is to animate visual nodes in your application. JavaFX also contains many builder classes that provide the ability to express a UI in a declarative style. Plus, JavaFX has a powerful property binding capability in which properties may be bound to expressions to automatically keep them updated. `</article>`

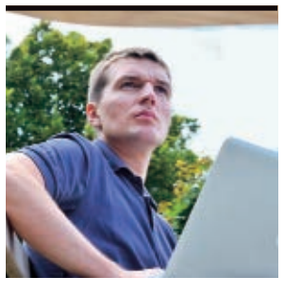
LEARN MORE

- JavaFX Technology at a Glance
- JavaFX Documentation

LISTING 5 LISTING 6 / LISTING 7 / LISTING 8

```
String playText = "Play";
...
StringProperty playButtonText =
  new SimpleStringProperty(playText);
...
playButton.textProperty()
  .bind(playButtonText);
```

See all listings as text



ADAM BIEN

BIO

Stress Testing Java EE 6 Applications

Use stress testing to identify application server configuration problems, potential bottlenecks, synchronization bugs, and memory leaks in Java EE 6 code.

Unit and integration tests are helpful for the identification of business logic bugs, but in the context of Java Platform, Enterprise Edition 6 (Java EE 6), they are meaningless. Both integration and unit tests access your application in a single-threaded way. After the deployment, however, your code will always be executed concurrently.

Stop Talking, Start Stressing

It is impossible to predict all nontrivial bottlenecks, deadlocks, and potential memory leaks by having theoretical discussions. It is also impossible to find memory leaks with unit and integration tests. Bottlenecks are caused by locks and I/O problems that are hard to identify in a single-threaded scenario. With lots of luck and patience, memory leaks can be identified with an integration test, but they can be far more easily spotted under massive load. The heavier the load, the greater is the probability of spotting potential concurrency and robustness problems. Cache behavior, the frequency of Java Persistence API (JPA) `OptimisticLockException`, and the amount of memory needed in production can also be evaluated easily with stress tests.

Even in the unlikely case of a perfect application without defects, your application server will typically be unable to handle the load with default factory settings. Stress tests are a perfect tool to learn the behavior of your application under load in the first iterations without any stress. Stress-test-driven development is the right choice for Java EE. Instead of applying optimizations prematurely, you should concentrate on implementing pure business logic and verifying the expected behavior with automated tests and hard numbers.

BUG SPOTTER

Java EE 6 applications are always executed concurrently. Even with unrealistic stress tests, you will learn a lot about **system behavior** and identify some bottlenecks or concurrency bugs.

Don't Be Realistic

Load tests are configured by taking into account the expected number of concurrent users and realistic user behavior. To meet the requirements, "think times" need to be kept realistic, which in turn reduces the amount of concurrency in the system. The heavier the load, the easier it is to find problems. Realistic load tests are usually performed too late in the development cycle and are useful only for ensuring that nonfunctional requirements are met. They are less valuable in verifying system correctness.

Instead of relying on realistic but lax load tests defined by domain experts, we should realize as much load as possible using developer-driven stress tests. The goal is not to verify expected scalability or performance. Instead, it is to find bugs and learn about the behavior of the system under load.

Stressing the Oracle

My "oracle" application records predictions and returns them as a JavaScript Object Notation (JSON) string. (For more information on the "oracle" application, see "Unit Testing for Java EE.")

Sending a GET request to the URI http://localhost:8080/oracle/resources/predictions returns a `Prediction` entity serialized as `{"prediction":{"result":"JAVA_IS_DEAD","predictionDate":"1970-01-01T19:57:39+01:00","success":"false"}}`. Services provided by Java API for RESTful Web Services (JAX-RS) are easily stress-testable; you need only execute several HTTP requests concurrently.

PHOTOGRAPH BY THOMAS EINBERGER/ GETTY IMAGES

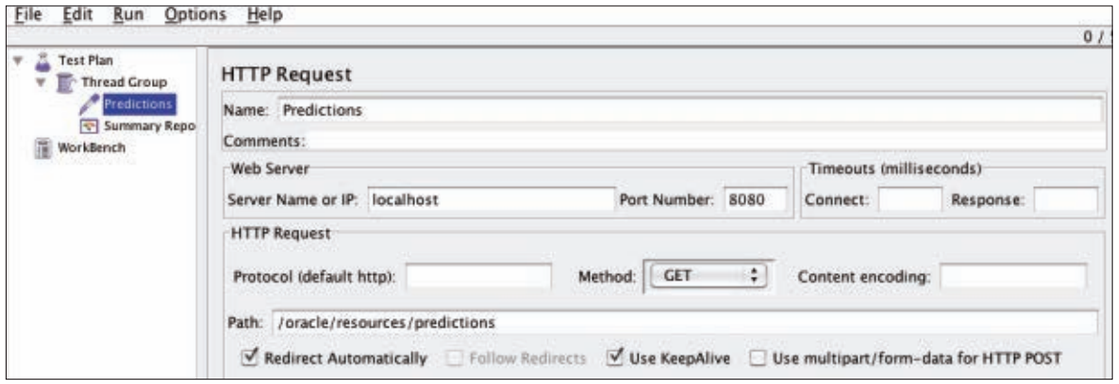


Figure 1

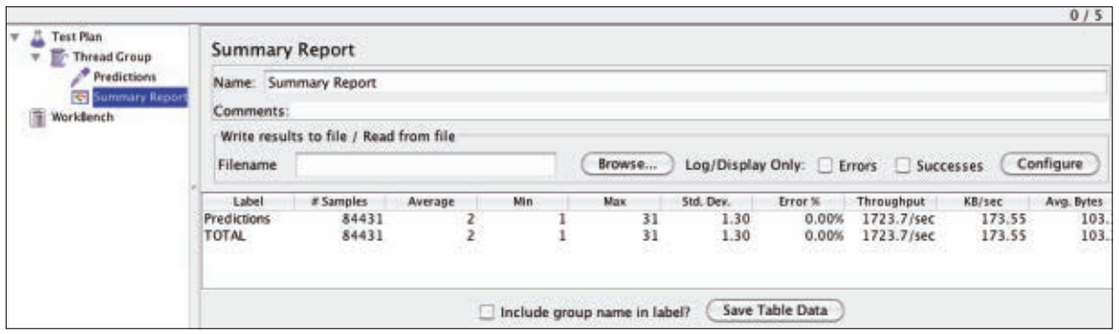


Figure 2

The open source load testing tool Apache JMeter comes with built-in HTTP support. After creating the ThreadGroup and setting the number of threads (and, thus, concurrent users), an HTTP request has to be configured to execute the GET requests (right-click, select **Sampler**, and then select **HTTP Request**). See **Figure 1**.

While the results can be visualized in various ways, the JMeter Summary Report is a good start (see **Figure 2**). It turns out that the sample application is able to handle 1,700 transactions per second for five concurrent users out of the box.

Every request is a true transaction and is processed by an Enterprise JavaBeans 3.1 (EJB 3.1) JAX-RS PredictionArchiveResource, delegated to the PredictionAudit EJB 3.1 bean, which in turn accesses the database through EntityManager (with exactly one record).

At this point, we have learned only that with EJB 3.1, JPA 2, and JAX-RS, we can achieve 1,700 transactions per second without any optimization. But we still have no idea what is happening under the hood.

VisualVM Turns Night to Day

GlassFish Server Open Source Edition 3.1.x and Java DB (the open source version of Apache Derby) are Java processes that can be easily monitored with VisualVM. Although VisualVM is shipped with the current JDK, you should check the VisualVM Website for updates.

VisualVM is able to connect locally or remotely to a Java process and monitor it. VisualVM provides an overview showing CPU load, memory consumption, number of loaded classes, and number of threads, as shown in **Figure 3**.

The overview is great for estimating resource consumption and monitoring the overall stability of the system. We learn from **Figure 3** that for 1,700 transactions per second, GlassFish Server Open Source Edition 3.1 needs 58 MB for the heap, 67 threads, and about 50 percent of the CPU. The other 50 percent was consumed by the load generator (JMeter).

Although this colocation is adequate for the purposes of this article, it blurs the results. The load generator should run on a dedicated machine or at least in an isolated (virtual) environment. Sometimes, you even have to run distributed JMeter load tests to generate enough load to stress the server. For internet applications, it might be necessary to deploy the load generators into the cloud.

In a stress test scenario, the plain numbers are interesting but unimportant. Stress tests do not generate a realistic load, but rather they try to break the system. To ensure stability, you should monitor the VisualVM Overview

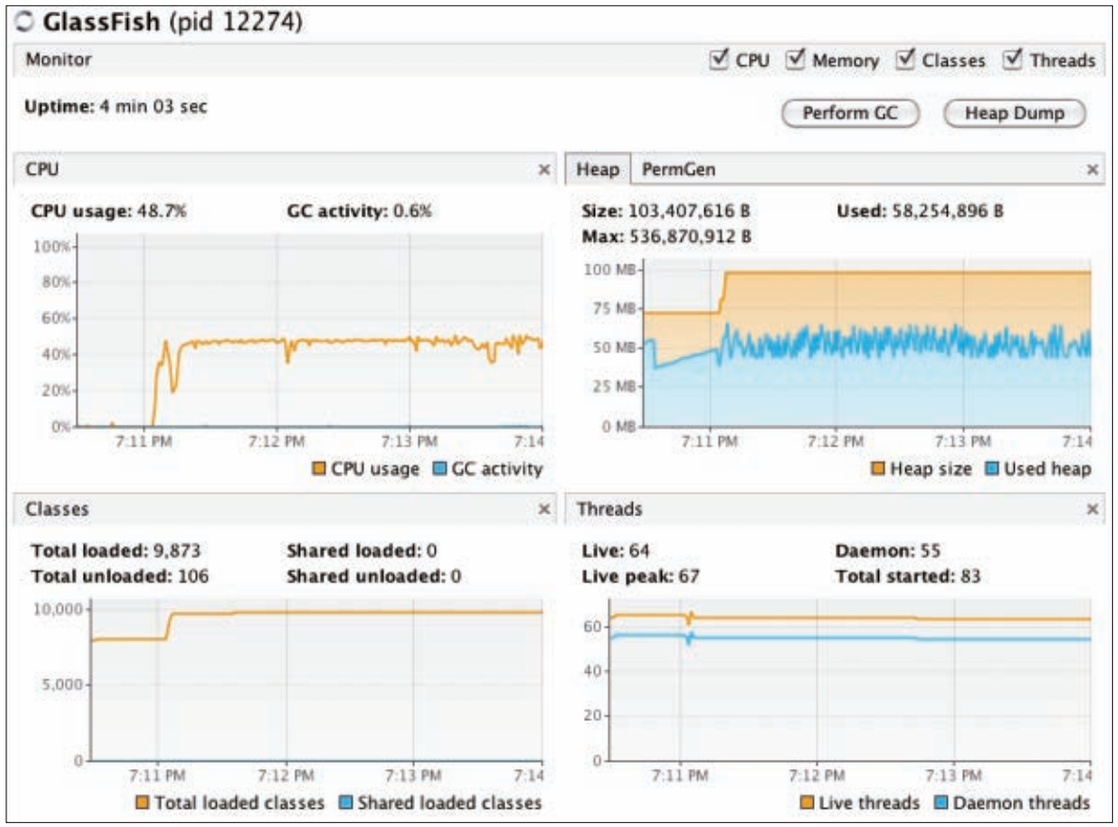


Figure 3

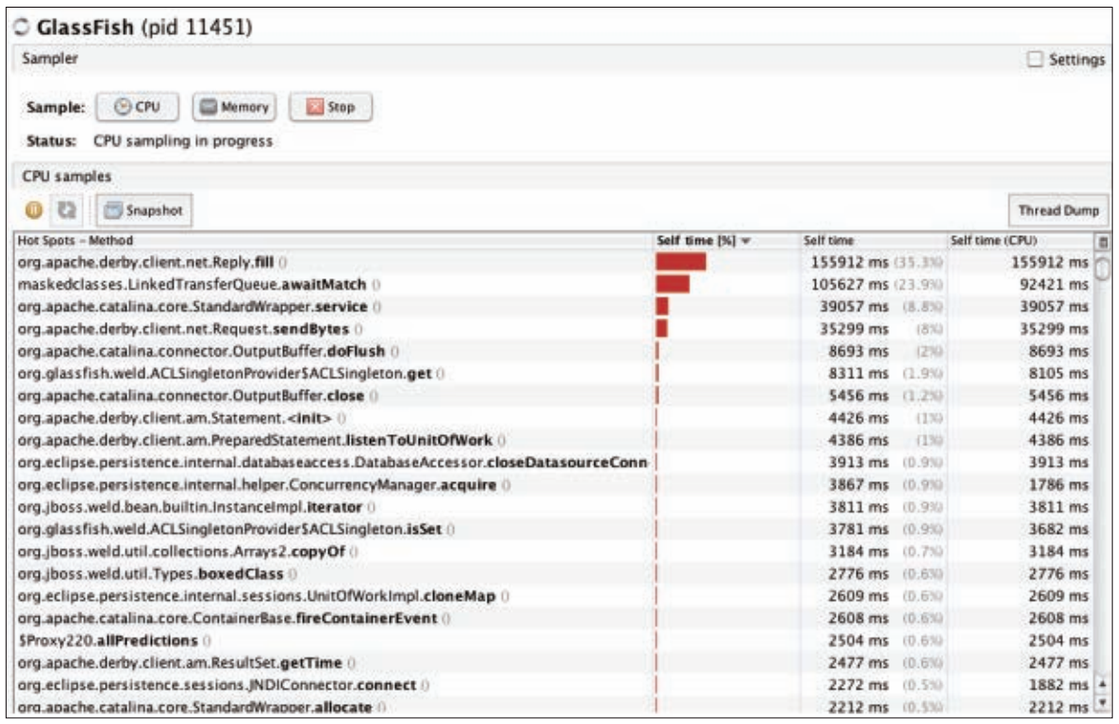


Figure 4

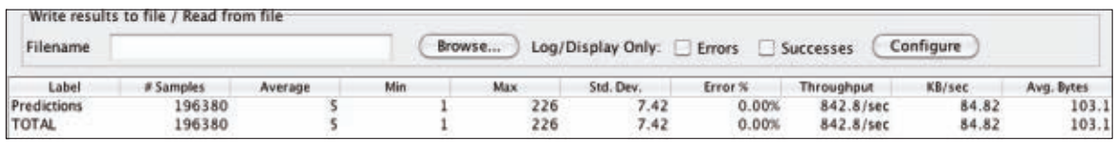


Figure 5

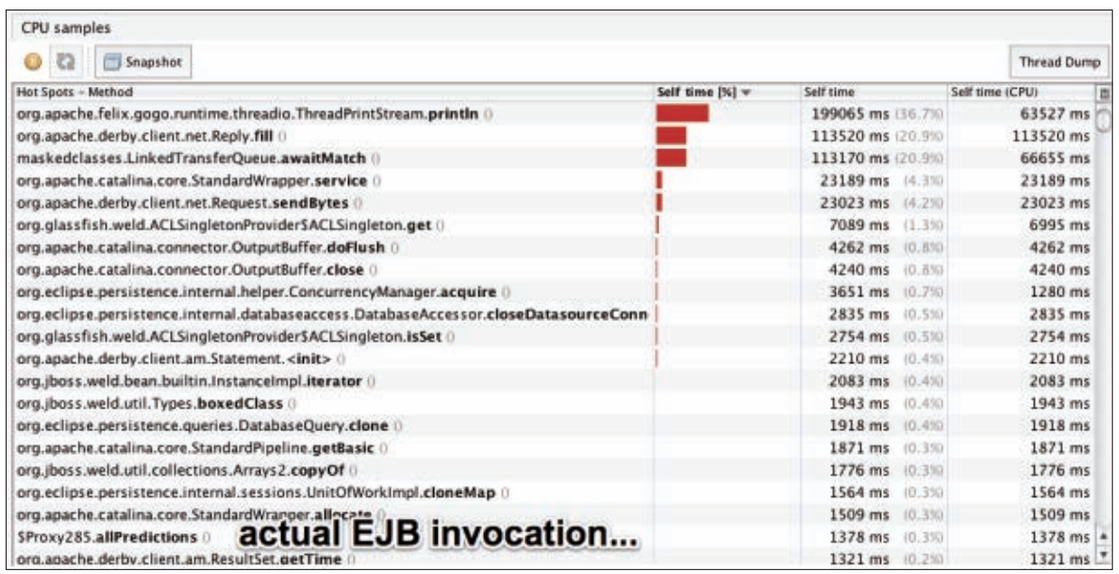


Figure 6

LISTING 1

```
public List<Prediction> allPredictions(){
  System.out.println("-- returning predictions");
  return this.em.createNamedQuery(Prediction.findAll).getResultList();
}
```

See all listings as text

average values. All the lines should be, on average, flat.

An increasing number of loaded classes might indicate problems with class loading and can lead to an OutOfMemoryError due to a shortage of PermGen space. An increasing number of threads indicates slow, asynchronous methods. A ThreadPool configured with an unbounded number of threads will also lead to an OutOfMemoryError. And a steady increase in memory consumption can eventually lead to an OutOfMemoryError caused by memory leaks.

VisualVM comes with an interesting profiling tool called Sampler. You can attach and detach to a running Java process with a little overhead and measure the most-expensive invocations or the size of objects (see **Figure 4**).

The sampling overhead is about 20 percent, so with an active sampler, you can still achieve 1,400 transactions per second. As expected, the application spends the largest amount of time communicating with the database.

How Expensive Is System.out.println?

A single System.out.println can lead to significant performance degradation. To measure the overhead, every invocation of the method allPredictions is logged with a System.out.println invocation, as shown in **Listing 1**.

Instead of 1,700 transactions per second, we are able to perform only about 800 transactions per second, as shown in **Figure 5**.

Let's take a look at the VisualVM Sampler output shown in **Figure 6**. More time is spent in ThreadPrintStream .println() than in the most expensive database operation.

The actual EJB 3.1 overhead is negligible. The $Proxy285.allPredictions() invocation is in the very last position and orders of magnitude faster than a single System.out.println.

Having a reference measurement makes identification of potential bottlenecks easy. You should perform stress tests as often as possible and compare the results. Performing nightly stress tests from the very first iteration is desirable. You will get fresh results each morning so you can start fixing potential bottlenecks.

Causing More Trouble

Misconfigured application servers are a common cause of bottlenecks. GlassFish

Server Open Source Edition 3.1 comes with reasonable settings, so we can reduce the maximum number of connections from the Derby pool to two connections to simulate a bottleneck. With five concurrent threads (users) and only two database connections, there should be some contention (see **Figure 7**).

The performance is still surprisingly good. We get 1,400 transactions per second with two connections. The max response time went up to 60 seconds, which correlates surprisingly well with the "Max Wait Time: 60000 ms" connection pool setting in GlassFish Server Open Source Edition 3.1. A hint in the log files also points to the problem, as shown in **Listing 2**.

Also interesting is the Sampler view in VisualVM (see **Figure 8**).

The method JNDIConnector.connect() became the most expensive method. It even displaced the Reply.fill() method from its first rank.

The package org.eclipse.persistence is the JPA provider for GlassFish Server Open Source Edition 3.1, so it should give us a hint about the bottleneck's location. There is nothing wrong with the persistence layer; it only has to wait for a free connection. This contention is caused by the artificial limitation of having only two connections available for five virtual users.

A look at the JNDIConnector.connect method confirms our suspicion (see **Listing 3**). In the method JNDIConnect .connect, a connection is acquired from a DataSource. In the case of an empty pool, the method will block until either an in-use connection becomes free or the Max Wait Time is reached. The method can block up to 60 seconds with the GlassFish Server Open Source Edition default settings. This rarely happens with the default settings, because the server ships with a Maximum Pool Size of 32 database connections.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
Predictions	290190	3	0	60028	258.55	0.00%	1398.2/sec	146.14	107.0
TOTAL	290190	3	0	60028	258.55	0.00%	1398.2/sec	146.14	107.0

Figure 7

CPU samples

Snapshot | Thread Dump

Hot Spots – Method	Self time [%]	Self time	Self time (CPU)
org.eclipse.persistence.sessions.JNDIConnector.**connect** ()		244018 ms (36.4%)	4463 ms
org.apache.derby.client.net.Reply.**fill** ()		126834 ms (18.9%)	126834 ms
maskedclasses.LinkedTransferQueue.**awaitMatch** ()		77974 ms (11.6%)	64153 ms
org.apache.catalina.core.StandardWrapper.**service** ()		51539 ms (7.7%)	51539 ms
org.apache.derby.client.net.Request.**sendBytes** ()		37897 ms (5.6%)	37897 ms
org.apache.catalina.connector.OutputBuffer.**doFlush** ()		13184 ms (2%)	13184 ms
org.glassfish.weld.ACLSingletonProvider$ACLSingleton.**get** ()		10831 ms (1.6%)	10735 ms
org.apache.catalina.connector.OutputBuffer.**close** ()		8955 ms (1.3%)	8955 ms
org.eclipse.persistence.internal.databaseaccess.DatabaseAccessor.**clos**		8310 ms (1.2%)	6551 ms
org.glassfish.weld.ACLSingletonProvider$ACLSingleton.**isSet** ()		5961 ms (0.9%)	5961 ms
org.apache.derby.client.am.Statement.**<init>** ()		5186 ms (0.8%)	5186 ms
org.eclipse.persistence.internal.helper.ConcurrencyManager.**acquire** ()		4921 ms (0.7%)	1885 ms
org.apache.derby.client.am.PreparedStatement.**listenToUnitOfWork** ()		4623 ms (0.7%)	4623 ms
$Proxy153.**name** ()		4491 ms (0.7%)	4491 ms
org.jboss.weld.util.collections.Arrays2.**copyOf** ()		4474 ms (0.7%)	4474 ms
org.eclipse.persistence.queries.DatabaseQuery.**clone** ()		4087 ms (0.6%)	4087 ms
org.eclipse.persistence.internal.sessions.UnitOfWorkImpl.**cloneMap** ()		3917 ms (0.6%)	3917 ms
org.apache.catalina.core.StandardWrapper.**allocate** ()		3146 ms (0.5%)	3146 ms

Figure 8

LISTING 2 LISTING 3

```
WARNING: RAR5117 : Failed to obtain/create connection from connection pool
[ SamplePool ]. Reason : com.sun.appserv.connectors.internal.api.PoolingException:
In-use connections equal max-pool-size and expired max-wait-time. Cannot allocate
more connections.
WARNING: RAR5114 : Error allocating connection : [Error in allocating a connection.
Cause: In-use connections equal max-pool-size and expired max-wait-time. Cannot
allocate more connections.]
```

See all listings as text

How to Get the Interesting Stuff

The combination of JMeter and VisualVM is useful for ad hoc measurements. In real-world projects, stress tests should be not only repeatable but also comparable. A history of results with visualization makes the resultant comparison and identification of hotspots easier.

VisualVM provides a good overview, but the really interesting monitoring information can be obtained only from an application server in a proprietary way. All major application servers provide extensive monitoring information via Java Management Extensions (JMX). GlassFish Server Open Source Edition 3.1 exposes its monitoring and management data through an easily accessible Representational State Transfer (REST) interface.

To activate the monitoring, open the GlassFish Admin Console by specifying the Admin Console URI (http://localhost:4848). Then select **Server**, select **Monitor**, and then select **Configure Monitoring**. Then select the **HIGH** level for all components. Alternatively, you can activate monitoring by using the asadmin command from the command line or by using the REST management interface.

Now, all the monitoring information is accessible from the following root URI: http://localhost:4848/monitoring/domain/server. The interface is self-explanatory. You can navigate through the components from a browser or from the command line.

The command curl -H "Accept: application/json" http://localhost:4848/monitoring/domain/server/jvm/memory/

usedheapsize-count returns the current heap size formatted as a JSON object (see **Listing 4**).

The most interesting key is usedheapsize-count. It contains the amount of used memory in bytes, as described by the description tag. The good news is that the entire monitoring API relies on the same structure and can be accessed in a generic way.

Monitoring Java EE 6 with Java EE 6

Executing HTTP GET requests from the command line still does not solve the challenge. To be comparable, the data has to be persistently stored. A periodic snapshot between 1 and 30 seconds is good enough for smoke tests and stress tests.

It turns out that you can easily persist monitoring data with a simple Java EE 6 application. JPA 2, EJB 3.1, Contexts and Dependency Injection (CDI), and JAX-RS reduce the task to only three classes. The JPA 2 entity Snapshot holds the relevant monitoring data (see **Listing 5**).

The entity Snapshot represents the interesting data, such as the number of busy threads, queuedConnections, errors, committed and rolled-back transactions, heap size, and the time stamp. You can extract such information from any other application server through different channels and APIs.

Listing 6 shows how to access REST services with Jersey. The managed bean DataProvider uses the Jersey client to access the GlassFish Server Open Source Edition's REST interface and convert the JSON result into Java primitives. The fetchData method is the core functionality of DataProvider, and it returns the populated Snapshot entity.

Every five seconds, the MonitoringController @Singleton EJB 3.1 bean shown in **Listing 7** asks the DataProvider for a Snapshot and persists it. In addition, the persisted data is exposed through REST. You can access all snapshots using http://localhost:8080/stm/resources/snapshots, and you will get Snapshot instances as a JSON object (see **Listing 8**).

Interestingly, a Java EE 6 solution is significantly leaner than a comparable Plain Old Java Object (POJO) implementation. Periodic timer execution, transactions, and EntityManager bookkeeping are provided out of the box in Java EE 6 but must be implemented in Java Platform, Standard Edition (Java SE).

Automating the Stress Test

Using the StressTestMonitor (STM) application, we can collect application server monitoring data systematically and persistently, and we can analyze and compare the stress test results after

GO FOR AVERAGE

In a **stress test scenario**, the plain numbers are interesting but unimportant. Stress tests do not generate a realistic load, but rather they try to break the system. To ensure **stability**, you should monitor the average values.

LISTING 4 LISTING 5 / LISTING 6 / LISTING 7 / LISTING 8

{"message":"","command":"Monitoring Data","exit_code":"SUCCESS",
"extraProperties":{"entity":{"**usedheapsize-count**":{"count":217666480,
"lastsampletime":1308569037982,"description":"Amount of used memory in
bytes","unit":"bytes","name":"UsedHeapSize","starttime":1308504654922}},
"childResources":{}}}.

See all listings as text

each run. This approach is not perfect, though, because both the stress test and the load generator must be started and stopped manually.

A continuous integration (CI) tool, such as Hudson or Jenkins, is capable of automating the whole lifecycle and can easily deploy STM and start the load generator automatically.

Hudson and Jenkins support a periodic build execution (for example, with a single configuration tag: @midnight). With Hudson or Jenkins, setting up a nightly executed stress test only takes minutes. You have to deploy the STM application first and launch the stress test generator afterward. For a GlassFish Server Open Source Edition deployment, this requires just a single line: asadmin deploy --force [...]/StressTestMonitor.war.

JMeter can also be started without the GUI in headless mode. It requires another line: jmeter -n -t predictions.jmx -l predictions.jtl. The parameter -n prevents the GUI from appearing, the parameter -t specifies the configuration file (created with JMeter in GUI mode), and -l specifies the log file, which can be analyzed with JMeter after the test.

A Hudson or Jenkins "Free Style Software Project" build with two build steps (Execute Windows Shell Command or Execute Shell) does the job: Deploy STM and Execute Stress Tests. You don't have to create a shell script or a batch file. The commands can be executed directly by Hudson or Jenkins.

VisualVM can still be used to monitor the application in real time or identify the hot spots. All the relevant monitoring data is persisted in a database table and can be analyzed easily after the test.

STRESS IS GOOD

Stress-test-driven development is the right choice for Java EE. Instead of applying optimizations prematurely, you should concentrate on **implementing pure business logic** and verifying the expected behavior with automated tests and hard numbers.

Nice-to-Haves

The solution described so far is good enough for getting started. The interesting parameters, such as JVM and application server monitoring data, are gathered and persisted automatically during a nightly job. Because of JSR-77 (the management and monitoring API), all application servers also provide access to the statistics of all deployed Java EE components. Application-specific EJB beans (usually the facade to your business logic) can be monitored using exactly the same mechanism.

You will get the number of concurrent requests, the current number of active instances (and, thus, the number of concurrent transactions accessing the bean), and business method runtime statistics, such as slowest and average execution times.

For example, PredictionAudit EJB bean pool data is accessible under http://localhost:4848/monitoring/domain/server/applications/com.abien_TestingEJBAndCDI.war_1.0-SNAPSHOT/PredictionAudit/bean-pool.

LISTING 9

```
@Inject
Event<Snapshot> escalationSink;
@Schedule(minute="*",second="*/5",hour="*")
public void gatherAndPersist(){
  Snapshot current = dataProvider.fetchData();
  em.persist(current);
  if(current.isSuspicious())
    escalationSink.fire(current);
}
```

See all listings as text

All the monitoring data is stored in a single table, which makes the data available to tools such as JasperReports or Eclipse BIRT. Charts and reports are only a few clicks away. The Snapshot entity is exposed through JSON, which makes it "consumable" by all JavaScript and JavaFX applications as well.

Furthermore, the Snapshot entity is a Java class and can contain additional validation or processing logic. It is trivial to escalate "suspicious" Snapshot instances with CDI events, as shown in **Listing 9**.

A suspicious Snapshot listener will only have to implement a method with an annotated parameter to receive the event public void onSuspiciousEvent(@Observes Snapshot snapshot){}.

All suspicious events can be analyzed in real time, sent using e-mail, or just aggregated and exposed with JMX.

Conclusion: No Excuses

Java EE 6 applications are always executed concurrently. Even with unrealistic stress tests, you will learn a lot about system behavior and identify some bottlenecks or concurrency bugs.

The earlier and more frequently stress tests are executed, the more you will learn about the application's runtime behavior. Application server configuration problems, potential bottlenecks, synchronization bugs, and memory leaks can also be identified during stress tests. </article>

LEARN MORE

- *Real World Java EE Night Hacks—Dissecting the Business Tier* (2011)
- Sample Java EE testing code on Project Kenai Web page
- StressTestMonitor application

VIKRAM GOYAL

BIO

Getting Started with GameCanvas and the Mobile Sensor API

Learn the concepts behind developing a location-aware game using the Mobile Sensor API.

As location-aware applications have become de rigueur, there has been an increasing demand to include location-based features in gaming as well. Most smartphones these days support the concepts behind location-aware games and applications, and Java is not behind in this respect. The Location API, JSR-179, provides all the nuts and bolts you need, so it is quite easy to incorporate location-based features in your games. For a quick overview of this API, see my article on gaming basics.

This article is the first article in a two-part series. In this article, I introduce the concepts behind developing a location-aware game, and I discuss how to use the Mobile Sensor API, JSR-256.

In Part 2, I will cover the Location API and explain the concepts required for combining location data with information from sensors to make a complete game.

This two-part series is not about developing a full-fledged game; rather, it introduces the concepts for how to combine location data with mobile sensors to make a game that can be the launching pad for a complete game. It introduces these concepts from a single player's perspective.

Note: The source code for the game I develop in this two-part series can be downloaded here.

The Game

The idea behind the game that I develop in this two-part series is simple. There is a red ball on the screen (see **Figure 1**), and you, as the first player, have to roll the ball left or right. Your friend, who has a compatible phone and game, has to "catch" the ball as it falls off your screen.

Two things have to be done to enable this. First, your gaming MIDlet needs to know if there are other players near you who have the capability of being a catcher, while you announce your availability. Second, you need to roll the ball using the accelerometer sensor on your phone and broadcast that to the other player so that her phone can catch it.

Understanding Gaming

From the simplest coding perspective, gaming is the art of repeatedly executing the same piece of code and updating screen variables depending on the user's (and game engine) input. A loop for repeatedly executing code is required in order to do the same thing over and over again until something in the gaming environment changes, due to either an external event (a player quits, for example) or an internal event (a time-out interval, for example). The loop checks an instance variable to make sure that none of those events has happened, and if none has, the repeated execution continues.

The loop checks for game inputs from the player and the environment and responds to them by updating its own internal variables and the game state. The game state is reflected on the screen, and the screen is thus updated. The loop pauses for a predetermined interval to let the game refresh itself.

The Java Platform, Micro Edition (Java ME) API provides five classes that help with the development of games: GameCanvas, Layer, Sprite, TiledLayer, and LayerManager. All these classes are in the javax.microedition.lcdui.game package and provide attributes and properties over and above the normal classes.

Figure 1

PHOTOGRAPH BY JONATHAN WOOD/ GETTY IMAGES

In this article, I explain the details of the GameCanvas class, because that is the only class we use for our example code. For more details on gaming in general and on the other classes, refer to my gaming basics article.

The GameCanvas Class

The Java ME API recognizes that a lot of MIDlets will be game-centric and, therefore, it provides the GameCanvas class. This class does everything that the normal Canvas class does, but it is extended to provide game-specific attributes, such as an offscreen graphics buffer and the ability to query key status.

The need for an offscreen buffer. Why is having an offscreen buffer important? Because it is an implementation of the double-buffering philosophy that is used to minimize flickering while rendering images in a game on the graphics display. If there were no offscreen buffer, the game would suffer from flicker issues, which would make it difficult to play the game.

When displaying content on a device's screen, the CPU process that is required for drawing the screen is usually intensive. Hence, if the drawing operation is done very quickly, the CPU lags. This lag produces a flicker effect.

To minimize this flicker, the drawing operation is done in an offscreen buffer. This buffer is in memory, and it is separate from the onscreen buffer. Once the drawing to this offscreen buffer is completed, this buffer can simply be copied across to the actual onscreen buffer. The copy operation is not as expensive as the direct rendering operation; hence, the load on the CPU is not as intense. This process minimizes the flicker that might be produced as the CPU completes the copy.

Offscreen buffer in GameCanvas. By providing an automatic offscreen buffer, the GameCanvas class minimizes the work that is required by the developer to maintain this buffer. This buffer is the same size as the device's display area. The area that should be copied across can be limited so that only parts of the actual screen are modified. This process of copying from the offscreen buffer to the main screen is usually called *flushing*. Flushing is implemented in the GameCanvas class by using the flushGraphics() method. Flushing doesn't clean this offscreen buffer; it simply copies the contents of the buffer to the device screen. Therefore, it is your responsibility to clean this buffer before starting to do another rendering operation; otherwise, the new rendering operation will mix with or override the previous render.

WHAT GAMERS WANT

As location-aware applications have become de rigueur, there has been an increasing demand to include **location-based features in gaming** as well.

Access to the actual graphics element is provided by the getGraphics() method of the GameCanvas class, which returns a handle to the Graphics object for the offscreen buffer. Each call to getGraphics() returns a new object, so it is prudent to reuse the Graphics object; however, each such newly returned object writes to the same offscreen buffer (as mentioned before, each GameCanvas gets its own buffer).

You can do all normal operations with this newly returned Graphics object. You can draw rectangles, images, and lines. All simple 2-D rendering operations can be implemented. The actual Graphics object has some default characteristics such as color (black), font (default font), stroke style (solid), and so on. You can change all of them to suit before rendering is done.

Game Workflow

To help you conceptualize the Mobile Sensor API use of this game, let's think of the workflow for getting this game started. Writing down the workflow of the game should help us design and code it.

1. Player 1 starts the game, and the game automatically searches for the current location in terms of GPS coordinates. If the location is not available, the phone displays a message and the player cannot play.
2. Once the location is found using the specified criteria, the game notifies the server about the coordinates. The server registers the coordinates using the phone's International Mobile Equipment Identity (IMEI) number, latitude, and longitude (and, optionally, the name of Player 1 as well).
3. When Player 1 is ready to play a game, Player 1 presses the Find button. The game confirms that a current location for Player 1 is present and then sends a request to the server to find players around that location.
4. The server returns a list of players, along with some unique identifiers.
5. Player 1 selects a fellow player and requests that the server start the game with the selected player.
6. Once Player 1 has requested that a game with a selected player be started, the game should notify the central server and initiate the loading of the game on Player 2's device.
7. After the initial handshake, Player 1's screen should show the ball in the bottom middle of the screen.
8. Then Player 1 can turn his phone left or right to move the ball.
9. Behind the scenes, the game should engage the tilt accelerometer and gather data from it.
10. If Player 1 is tilting the phone left along the *y* axis, the ball should move left in the *x* axis (relative to its position on the phone screen). Similarly, if Player 1 tilts the phone right along the *y* axis, the ball should move right in the *x* axis (relative to its position on the phone screen).
11. Once the ball reaches the boundary conditions of 0 or screen width, the server should be notified, and the waiting Player 2's screen should be located.
12. If Player 2 is at the same location and has tilted the phone at the same tilt, she should be able to "catch" the ball. The server should indicate success or failure accordingly.

To keep things simple, in the following sections I will show you the interactions only from Player 1's perspective. The server and Player 2 interactions are easy to code and conceptualize, so I will leave these as an exercise for you. First things first: we need to create our GameCanvas and add the code for the Mobile Sensor API.

The TiltCanvas Class

The TiltCanvas class is responsible for handling game interactions and implementing the Mobile Sensor API code. **Listing 1** shows the constructor.

The TiltSensor constructor does the initialization by opening a connection to the sensor for the accelerometer, and it also loads the basic ball image. Note that it makes a call to the superconstructor because TiltCanvas extends GameCanvas.

```
class TiltCanvas extends
GameCanvas implements
DataListener, Runnable
```

We also need to implement the DataListener interface of the Mobile Sensor API to receive data when the device is tilted. Besides this, the Runnable interface is implemented to run the code in a thread.

Mobile Sensor API Implementation

In our game, we want a particular type of sensor. This sensor is universally known as the *acceleration sensor*. So, in the constructor of our TiltCanvas class we find out whether the device has this sensor, and if it does, we determine whether we can connect to it and receive data from it, as shown in **Listing 2**.

If the sensor is not found, then the game cannot be played, and the player should be informed. When it is found, we make a connection to it, and set the TiltCanvas class as the listener for receiving the data from it.

The sensor sends the current title of the device along the three axes: *x*, *y*, and *z*. Because we only want to know how the player has tilted the device along the vertical axis, we use only the *y* axis data. So, if the player tilted the device more than the previous tilt, the ball is rolled to the right. On the other hand, if the player tilted the device less than the previous tilt, the ball is rolled to the left. Once we have reached the end of the screen, either along the left or right side, we notify the server with the current data.

There are game specifics in this listing that I haven't discussed yet. **Listing 3** shows the corresponding code for this class that is responsible for drawing the ball depending on the location data it gets from the dataReceived method.

This is a very standard method for a gaming application. The screen is cleared, and based on the current game data, objects on the screen are redrawn. In our case, the current position of the ball is drawn on the screen along the *x* axis, because the *y* axis is presumed to be constant.

Conclusion

In this article, we discussed the basics of gaming and the need for the GameCanvas class. We also saw how to incorporate the Mobile Sensor API into the specific game we were building.

In the next article, we will learn how to incorporate the Location API into our game. `</article>`

LISTING 1 LISTING 2 / LISTING 3

```
// the graphics object for this screen
Graphics g;

public TiltCanvas() {

 super(true);

 SensorInfo sensors[] =
  SensorManager.findSensors(
  "acceleration", SensorInfo.CONTEXT_TYPE_DEVICE);

 if (sensors == null || sensors.length == 0) {
  System.err.println("Nothing found!");
 } else if (sensors.length > 1) {
  System.err.println("Too many sensors found!");
  } else {
  tiltSensor = sensors[0];
  try {

   // make a connection to this sensor
   connection = (SensorConnection) Connector.open(tiltSensor.getUrl()); connection.
setDataListener(this, 10);

   // load the ball image
   ballImage = Image.createImage("/red_ball.jpg");

   setupCorrect = true;
   } catch (IOException ex) {
   ex.printStackTrace();
  }
 }

}
```

See all listings as text

LEARN MORE

- "J2ME Tutorial, Part 3: Exploring the Game API of MIDP 2.0"
- "Working with the Mobile Sensor API"

BENJAMIN J. EVANS AND **MARTIJN** VERBURG

BIO

Polyglot Programming on the JVM

Tips about how and why to choose a non-Java language for your project

The following is an excerpt from The Well-Grounded Java Developer, *which Manning Publications is due to release in January 2012.*

The phrase *polyglot programming on the JVM* is relatively new. It was coined to describe projects that utilize one or more non-Java Java Virtual Machine (JVM) languages alongside a core of Java code. One common way to think about polyglot programming is as a form of separation of concerns. As you can see in **Figure 1**, there are potentially three layers where non-Java technologies can play a useful role. This diagram is sometimes called the polyglot programming pyramid and comes from the work of Ola Bini.

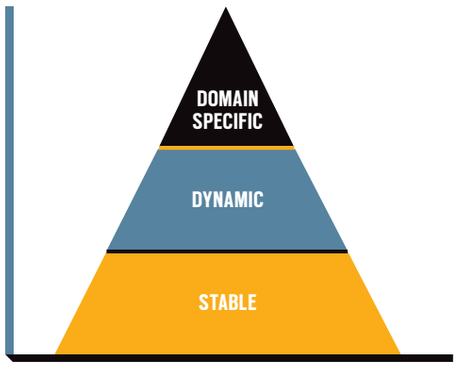


Figure 1

Within the pyramid, you can see three well-defined layers: domain-specific, dynamic, and stable. **Table 1** shows these three layers in more detail.

As you can see, there are patterns in the layers—the statically typed languages tend to gravitate toward tasks in the stable layer. Conversely, the less powerful and more specific-purpose technologies tend to be suited to domain roles at the top of the pyramid.

In the middle of the pyramid, we see that there is a rich role for languages in the dynamic tier. These languages are also the most flexible; in many cases, overlap might exist between the dynamic tier and either of the neighboring tiers.

Let's dig a little deeper into this diagram and look at why Java isn't the best choice for everything on the pyramid.

Why Use a Non-Java Language?

Java's nature as a general-purpose, statically typed, compiled language provides many advantages. These qualities make the language a great choice for implementing functionality in the stable layer.

LAYER	DESCRIPTION	EXAMPLES
DOMAIN-SPECIFIC	DOMAIN-SPECIFIC LANGUAGE (DSL); TIGHTLY COUPLED TO A SPECIFIC PART OF THE APPLICATION DOMAIN	APACHE CAMEL DSL, DROOLS, WEB TEMPLATING
DYNAMIC	RAPID, PRODUCTIVE, FLEXIBLE DEVELOPMENT OF FUNCTIONALITY	GROOVY, JYTHON, CLOJURE
STABLE	CORE FUNCTIONALITY, STABLE, WELL-TESTED, PERFORMANT	JAVA, SCALA

Table 1

However, these same attributes become a burden in the middle and upper tiers of the pyramid; for example:

- Recompilation is laborious.
- Static typing can be inflexible and lead to long refactoring times.
- Deployment is a heavyweight process.
- Java's syntax is not a natural fit for producing DSLs.

A pragmatic solution is to play to Java's strengths and take advantage of its rich API and library support to do the heavy lifting for the application—down in the stable layer.

If you're starting a new project from scratch, you might also find that another stable layer language (for example, Scala) has a particular feature (for example, superior concurrency support) that is important to your project. In most cases, however, you should not throw out working stable layer code and rewrite it in a different stable language.

At this point, you may be asking yourself, "What type of programming challenges fit inside these layers? Which languages should I choose?" A well-grounded Java developer knows that there is no silver bullet, but there are some criteria you can consider when evaluating your choices.

How to Choose a Language for Your Project

Once you've decided to experiment with non-Java approaches in your project, you need to iden-

PHOTOGRAPHY BY JOHN BLYTHE AND BOB ADLER

tify which parts of your project best suit a dynamic layer or domain-specific layer approach. **Table 2** highlights some tasks that might be suitable for each layer.

There is a wide range of natural use cases for alternative languages. However, identifying a task that could be accomplished with an alternative language is just the beginning. You now need to evaluate whether using an alternative language is appropriate. Here are some useful criteria that we take into account when considering technology stacks:

- Is the project area low risk?
- How easily does the language interoperate with Java?
- What tooling and testing support is available for the language?
- How easy is it to compile, test, build, and deploy software in this language?
- How difficult is the learning curve?
- How easy or difficult is it to hire developers with experience in this language?

Let's dive into each of these areas so you get an idea of the sorts of questions you need to be asking yourself.

Is the Project Area Low Risk?

Let's say you have a core, payment-processing rules engine that handles more than 1 million transactions a day. It is a stable piece of Java software that has been around for more than seven years, but there are not many tests for it and there are plenty of dark corners in the code. The core of this engine is clearly a high-risk area for a new language, especially because it is running successfully and there is a lack of test coverage and a shortage of developers who fully understand it.

However, there is more to a system than just its core processing. For example, this is a situation where better tests would clearly help. Scala has a great testing framework called ScalaTest. This enables developers to produce JUnit-like tests for Java code but without a lot of the boilerplate that JUnit seems to generate. So, once they are over the initial learning curve, developers can be much more productive at improving the test coverage. ScalaTest also provides a great way to gradually introduce concepts such as Behavior-Driven Development to the codebase. These concepts can really help when the time comes to refactor or replace parts of the core—regardless of whether the new processing engine ends up being written in Java or Scala.

Or, suppose that the operations users would like to have a Web console built so they can administer some of the noncritical, static data behind the payment-processing system. The developers already know Struts and JavaServer Faces, but they don't feel any enthusiasm for either technology. This is another low-risk area to try out a new language and technology stack. One obvious choice would be Grails. Developer buzz, backed up by some studies (including one by Matt Raible), is that Grails is the best available Web framework for productivity.

By focusing on a limited pilot in a low-risk area, the manager always has the option of terminating the project and porting to a different delivery technology without too much disruption, if it turns out that the attempted technology stack was not a good fit for the team or system.

Does It Interoperate with Java?

You don't want to lose the value of all that great Java code you've already written. This is one of the main reasons organizations are hesitant to introduce a new programming language into their technology stack. However, with alternative languages that run on the JVM, you can turn this concern on its head—so it becomes about maximizing the existing value in your codebase, not about throwing away working code.

Alternative languages on the JVM are able to cleanly interoperate with Java and can, of course, be deployed in a pre-existing environment. This is especially important when discussing this step with the production management folks. By using a non-Java JVM language as part of your system, you'll be able to make use of their expertise in supporting the existing environment. This can help reduce risk and alleviate any worries they might have about the new solution.

Note: DSLs are typically built using a dynamic (or, in some cases, stable) layer language, so many of them run on the JVM via the languages they were built in.

Some languages interoperate with Java more easily than others. We've found that most popular JVM alternatives have good interoperability with Java. Run a few experiments first to really make sure you understand how the integration can work for you.

Take Groovy, for example. You can simply import Java packages directly into its code using the familiar `import` statement. You can build a quick Website using the Groovy-based Grails framework and yet still reference your Java model objects. Conversely, it's very easy for Java to call Groovy code in a variety of ways and to receive familiar Java objects. One example use case could be calling out to Groovy from Java to process some JSON and having a Java object be returned.

LAYER	EXAMPLE PROBLEM DOMAINS
DOMAIN-SPECIFIC	BUILD, CONTINUOUS INTEGRATION, AND CONTINUOUS DEPLOYMENT DEV-OPS ENTERPRISE INTEGRATION PATTERN (EIP) MODELING BUSINESS RULES MODELING
DYNAMIC	RAPID WEB DEVELOPMENT PROTOTYPING INTERACTIVE ADMINISTRATIVE/USER CONSOLES SCRIPTING TEST-DRIVEN DEVELOPMENT (TDD) AND BEHAVIOR-DRIVEN DEVELOPMENT (BDD)
STABLE	CONCURRENT CODE APPLICATION CONTAINERS CORE BUSINESS FUNCTIONALITY

Table 2

Is There Tooling and Test Support?

Most developers underestimate the amount of time they save once they become comfortable in their environment. Their powerful IDEs, build tools, and test tools help them rapidly produce high-quality software. Java developers have benefited from great tooling support for years now, so it's important to remember that other languages might not be at quite the same level of maturity.

Some languages (for example, Groovy) have had long-standing IDE support for compiling, testing, and deploying the end result. Other languages might have tooling that is not as fully matured yet. For example, Scala's IDEs are not as polished as those of Java, but Scala fans feel that the power and conciseness of Scala more than make up for the imperfections of the current generation of IDEs.

A related issue is that when an alternative language has developed a powerful tool for its own use (such as Clojure's awesome Leiningen build tool), the tool might not be well adapted to handle other languages. This means that the team needs to think carefully about how to divide up a project, especially for deployment of separate but related components.

WHEN TO EVALUATE

There is a wide range of natural use cases for **alternative languages**. However, identifying a task that could be accomplished with an alternative language is just the beginning. You **now need to evaluate** whether using an alternative language is appropriate.

Is It Difficult to Learn?

It always takes time to learn a new language, and that time only increases if the paradigm of the language is not one that your development team is familiar with. Most Java development teams will be comfortable picking up a new language if it is object-oriented with a C-like syntax.

It gets harder for Java developers as they move further away from this paradigm. Scala tries to bridge the gap between the object-oriented and the functional worlds, but the jury is still out on whether this fusion is viable for large-scale software projects. At the extreme end of the popular alternative languages, a language such as Clojure can bring incredibly powerful benefits but can also represent a significant retraining requirement for development teams.

One alternative is to look at the JVM languages that are reimplementations of existing languages. Ruby and Python are well-established languages with plenty of available material with which developers can educate themselves. The JVM incarnations of these languages could provide a sweet spot for your teams to begin working with an easy-to-learn, non-Java language.

Do Developers Use the Language?

Organizations have to be pragmatic; they can't always hire the top two percent, and their development teams will change throughout the course of a year. Some languages, such as Groovy and Scala, are becoming established enough that there is a pool of developers to hire from. However, a language such as Clojure is still finding its way to popularity and, therefore, finding good Clojure developers is difficult.

Again, the reimplemented languages can potentially help here. Few developers might have JRuby on their résumés, but because it is just Ruby on the JVM there is actually a large pool of developers to hire from.

Note: One word of warning about the reimplemented languages: Many existing packages and applications written, for example, in Ruby, are tested only against the original, C-based implementation. This means that there might be problems when trying to use them on top of the JVM. When making platform decisions, you should factor in extra testing time if you are planning to leverage an entire stack written in a reimplemented language.

Summary

For the polyglot programmer, languages fall roughly into three programming layers: stable, dynamic, and domain-specific. Languages such as Java and Scala are best used for the stable layer of software development, while others, such as Groovy and Clojure, are more suited to tasks in the dynamic or domain-specific realms.

Certain programming challenges fit well into particular layers, for example, rapid Web development for the dynamic layer or modeling enterprise messaging for the domain-specific layer.

It's worth emphasizing again that the core business functionality of an existing production application is almost never the correct place to start when introducing a new language. The core is where high-grade support, excellent test coverage, and a proven track record of stability are paramount. Rather than start here, choose a low-risk area for the first deployment of an alternative language.

Finally, always remember that every team and project has its own unique characteristics that will affect the language choice. When choosing to implement a new language, managers and senior techs must consider the nature of their projects and team.

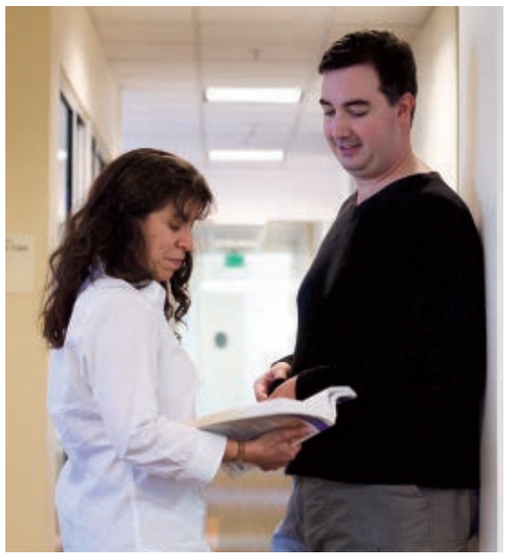
A small team composed exclusively of experienced propellerheads might choose Clojure for its clean design, sophistication, and power. Meanwhile, a Web shop that is looking to grow the team quickly and attract young developers might choose Groovy and Grails for the productivity gains and relatively deep talent pool. `</article>`

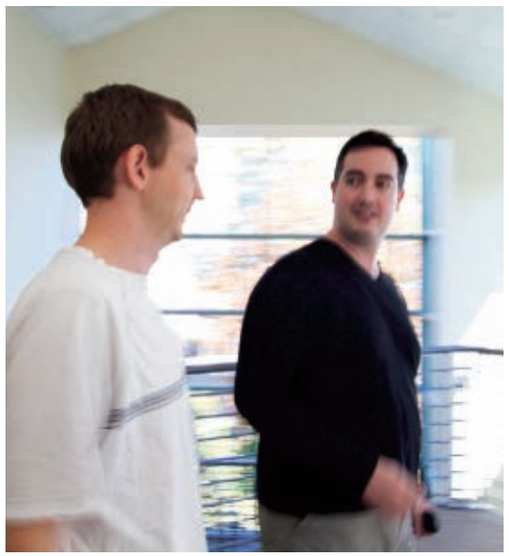
LEARN MORE

- *The Well-Grounded Java Developer*
- *Scala in Action* (Manning, 2010)
- *AspectJ in Action, Second Edition* (Manning, 2009)
- *DSLs in Action* (Manning, 2010)

JVM Language Summit—Toward a Universal VM

Oracle's **Alex Buckley** talks with *Java Magazine* about the 2011 JVM Language Summit and the evolution of the JVM and the Java platform. **BY MICHAEL MELOAN**





Alex Buckley, Specification Lead for the Java Language and JVM at Oracle

PHOTOGRAPHY BY BOB ADLER

The JVM Language Summit, which took place July 18–20, 2011, at Oracle's Santa Clara campus, explored the state of the art in language design and implementation on the Java Virtual Machine (JVM). It was an open collaboration among VM architects, runtime engineers, tool builders, compiler experts, and language designers. The Summit presented an opportunity for a wide range of participants to cross-pollinate technologies through presentations and conversations. Alex Buckley, specification lead for the Java language and JVM at Oracle, talked with Java Magazine *about the event and related themes.*

Java Magazine: What were some of the most memorable topics at this year's JVM Language Summit?

Buckley: There was a strong sense at this gathering that many of the technical directions that began at the first JVM Language Summit in 2008 are now coming to fruition, primarily due to the availability of the Java Platform, Standard Edition 7 (Java SE 7) invokedynamic instruction and method handles (JSR-292).

Language implementers talked extensively about how they use invokedynamic to improve speed and efficiency. JRuby, Jython, JavaScript (Oracle's Nashorn), Fortress (Oracle's scientific programming language), and even a Smalltalk variant (Rtalk) were all represented. Cameron Purdy from Oracle gave a fascinating presentation covering eclectic territory, including a wish list of JVM features going forward, such as tail recursion, tail call optimization, continuations, properties, class file extensions, and more-intrinsic types, such as unsigned integers.

Rémi Forax presented a cookbook of techniques for utilizing invokedynamic that are applicable to many languages. And Oracle's Brian Goetz talked about how to compile lambda expressions into JVM bytecode (coming in Java SE 8), in a way that doesn't generate extra class files and that gives the Java compiler a great deal of flexibility in how to represent lambda expressions at runtime. This is accomplished by using invokedynamic to set up the creation of the lambda expressions, which are simply blocks of code, a bit like

JVM FACT

In many cases, the **same tool suite** can be used on the JVM regardless of the source language.

a method, but without all the surrounding class structure.

Java Magazine: Will you give us an overview of some of the most important languages running on the JVM? And how might they be used with Java to improve developer productivity and create more-flexible and robust applications?

Buckley: Large systems today succeed based on their technical architecture. That architecture typically has many loosely coupled components, each fulfilling a very specific purpose. Typically, these components collaborate by sharing data in a relational database, or a message bus, or by exposing XML services to each other. The more loosely coupled this architecture, the easier it is to choose the best language for the different components.

User interfaces are generally Web-based and written in HTML or JavaScript, with middleware often written in Java, and back-end logic written in Java, Ruby, or Scala. Groovy is fairly common at build time, and server-side JavaScript is growing in popularity.

A well-designed architecture is clear about the functional components and how they interact. Then the appropriate language can be chosen for those components. Organizations are searching for the right mix of productivity and maintainability in the languages they use. Especially on the JVM, components can be swapped in and out over time. Sometimes a Java component is rewritten in Scala, and sometimes a JRuby component is rewritten in Java. In many cases, the same tool suite can be used on the JVM regardless of the source language.

Java Magazine: Oracle's upcoming Nashorn JavaScript interpreter was covered at the JVM Language Summit. Can you give us some details?

Buckley: Nashorn is an Oracle-led project currently scheduled to be available to developers in JDK 8. It's an implementation of JavaScript running on the JVM, targeted specifically for the benefit of Java developers. We want to expose JavaScript to Java collections and Enterprise JavaBeans. And we want to ensure that the boundary between Java and JavaScript is as thin and interoperable as possible.

The development of a JavaScript interpreter is partially motivated by the fact that much of the world is moving toward HTML5 for cross-platform GUIs. HTML5 offers many JavaScript APIs for accomplishing some very useful tasks, such as location sensing or database access in the browser. The scope of HTML5 has expanded far beyond "mark-up" for layout, and the natural way for these additional front-end services to be exposed is through JavaScript. Nashorn would allow a non-browser JavaScript program running on the client or server direct access to standard JavaScript APIs and the vast collection of Java APIs.

The use of invokedynamic and method handles makes implementing JavaScript, which is an extremely dynamic and weakly typed object-based language, on the JVM surprisingly straightforward.

Java Magazine: How will the invokedynamic instruction be directly useful to the Java language in Java SE 8?

Buckley: The implementation of lambda expressions (Project Lambda) is a good example. It's something of a challenge, because the JVM is conceptually streamlined and very focused. The JVM has no knowledge of a lambda expression, a closure, or a continuation. It has no knowledge of type inference or any other language feature that might be designed in support of lambda expressions.

For many years, we could add features to the Java language that mapped directly to JVM features, or in the case of generics, did not show up in the JVM at all. That era of direct mapping is over. With Project Lambda, we needed a way to represent lambda expressions on top of the JVM in a way that is efficient and extensible. In terms of disk space, memory usage, and other overhead, anonymous inner classes were not a reasonable solution.

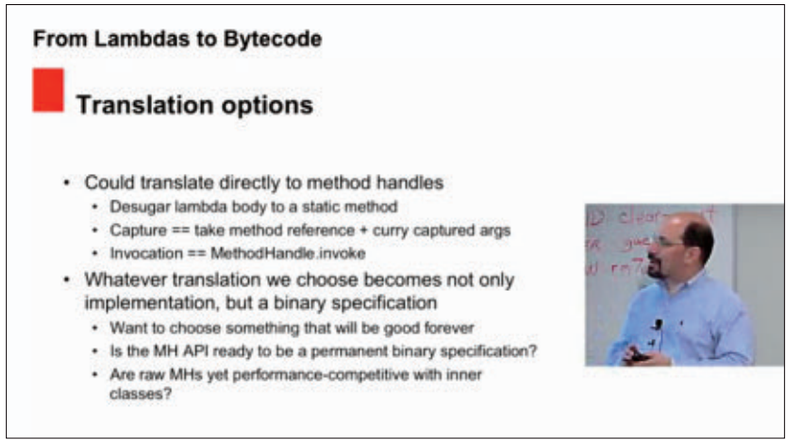
The invokedynamic instruction provided the solution. It essentially facilitates quick access to method handles. Method handles are really function pointers, and function pointers are a way of manipulating individual blocks of code, which is what lambda expressions are all about. The goal is to give the Java compiler a great deal of flexibility in how to represent these lambda expressions at runtime. That is accomplished by using invokedynamic to set up their creation.

Java Magazine: What are some examples of how lambda expressions could benefit Java developers in a tangible way?

Buckley: Lambda expressions are relevant with APIs. The bulk of what developers do involves utilizing someone else's APIs. A very common task in Java programming is iterating through a collection of business objects, finding ones that match some requirement, and passing that result on to the next step of the program.

Rather than extracting items and writing code to perform those operations (and hoping that nobody else is mutating the collection at the same time), using lambda expressions simplifies things dramatically. If you can pass code, in the form of a lambda expression, to a library, you are basically passing it to the collection itself. Then, concerns such as thread safety become the responsibility of the library implementer, rather than the responsibility of the calling programmer.

High-level operations improve software readability and performance.



Brian Goetz, *From Lambdas to Bytecode*

Listing 1, **Listing 2**, and **Listing 3** show an example of code simplification through the use of lambda expressions.

In **Listing 1**, existing collections impose *external iteration*. The client of the collection determines how to iterate. The implementation of accumulation is overspecified, and computation is achieved through side effects (assignment to highestScore).

Listing 2 uses *internal iteration*, so the iteration and accumulation are embodied in the library. For example, filtering can be done in parallel, and the client is more flexible, more abstract, and less error prone.

Listing 3 uses lambda expressions, which represent "code as data." The lambda expression is introduced with special syntax using zero or more-formal parameters. Parameter types are optional and may be inferred, and the body may be an expression or statements. If the body is an expression, there is no need for return or ;.

We refer to internal iteration when the collection is responsible for iterating or in some way processing its data. By passing the programming logic to the collection and letting the collection iterate through its elements, the program is potentially much more efficient and robust than with external iteration, where the iteration logic lives in the user's program.

Having lambda expressions in the language makes it much more natural to design and use "higher-order functions" in which the programmer specifies just the atomic operations to be performed on the data, and the collection figures out how to accomplish the task. Using this technique, many applications can be written in a way that is more compact and reliable. In a sense, lambda expressions are the fundamental primitive in a programming language, and now they can be implemented in an efficient manner.

Java Magazine: As specification lead for the JVM and the Java language, will you give us some insight into the process of Java evolution through specification?

Buckley: The Java Community Process [JCP] provides the framework for the evolution of the Java language, the core Java APIs, and the JVM. A Java Specification Request [JSR], such as JSR-292, that modifies the JVM ultimately becomes part of the Java SE Platform Specification.

It takes multiple years to make this kind of major change because we not only have to create the design space correctly, but we also need to allow time to specify things precisely enough to answer implementers' questions but generically enough to avoid implementation-specific artifacts in the design.

In other words, we don't specify the platform in terms of various software implementations; we implement in terms of specification. It's a much more structured process than most open source projects, though I think the worlds of open source and community standards are moving closer together.

It's important to have a deeply involved Expert Group like John Rose had for JSR-292. Expert Group members contribute their deep experience. For instance, a language implementer for JRuby might say, "My life would be easier if the JVM helped me do this . . . ." As a JSR progresses, external contributors also provide reports and the benefit of "in the trenches" observations.

I think JSR-292 did a remarkable job coming up with a design that satisfies language implementers and is fully implementable with proper performance by all the major JVM vendors. The design that was committed to will be part of the Java SE platform "forever," because while JVM implementations come and go, the JVM itself is a lasting entity. There will probably be class files in computing environments 50 years from now! At this level of the platform, everything must be impeccably stable and reliable.

The Java SE platform is evolving in a deliberate and mature way, through multiple implementations, the standardization process, and adoption by major language implementers. Some say that the ship moves rather slowly. But when a great ship finally leaves port, it's quite a sight. `</article>`

THE FUNDAMENTALS

In a sense, **lambda expressions** are the fundamental primitive in a programming language, and now they can be implemented in an **efficient** manner.

LISTING 1 LISTING 2 / LISTING 3

```
double highestScore = 0.0;
for (Student s : students) {
  if (s.gradYear == 2011) {
    if (s.score > highestScore) {
      highestScore = s.score; }
    }
}
```

See all listings as text

Michael Meloan began his professional career writing IBM mainframe and DEC PDP-11 assembly languages. He went on to code in PL/I, APL, C, and Java. In addition, his fiction has appeared in *WIRED*, *BUZZ*, *Chic*, *LA Weekly*, and on National Public Radio. He is also a Huffington Post blogger.

LEARN MORE

- JSR-292: "Supporting Dynamically Typed Languages on the Java Platform"
- Project Lambda
- Java SE

//fix this /

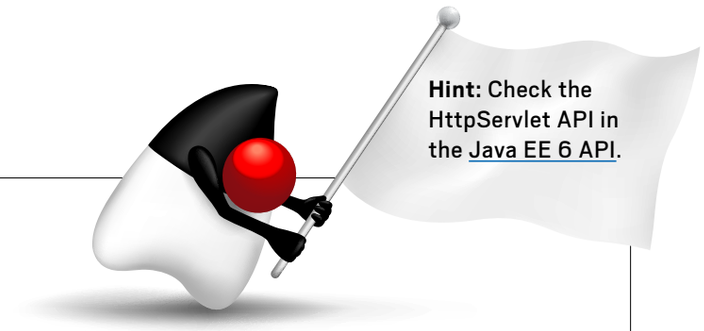
In the last issue, Arun Gupta, a Java evangelist at Oracle, posed a challenge about the CDI specification and asked readers for a fix.

The correct answer is #2: "beans.xml" is required to enable injection. The CDI specification requires "beans.xml" in the WEB-INF directory in order for bean injection to work. So even though the code is fine, a missing file did not allow the EJB injection to go through.

And now, for Gupta's latest challenge.

1 THE PROBLEM

Java EE 6 allows you to create Web applications very easily using annotations on POJOs. A POJO can be easily converted to a Servlet by adding @WebServlet and extending HttpServlet. A POJO becomes a JPA bean by adding @Entity. Each method of a Java class becomes implicitly transactional by adding @Stateless. Following the convention-over-configuration paradigm, the deployment descriptors are optional for most of the common cases.



Hint: Check the HttpServlet API in the Java EE 6 API.

2 THE CODE

Consider the following piece of code to have database access within a Servlet:

```
@WebServlet(name = "TestServlet", urlPatterns = {"/TestServlet"})
public class TestServlet extends HttpServlet {

   @PersistenceContext
   EntityManager em;

   protected void doGet(HttpServletRequest request,
HttpServletResponse response) {
      // Invoke methods on Entity Manager
   }
}
```

3 WHAT'S THE FIX?

Is this code guaranteed to work in a multithreaded environment?

1) Of course. All the samples are written this way.
2) No. Servlets are re-entrant, and EntityManager is not thread-safe. Instead inject as

   ```
   @PersistenceUnit EntityManagerFactory em;
   ```

   and then get EntityManager within the doGet() method.
3) Database access in Servlets must be done through EJBs only. The correct way is to move the database code in an EJB, inject the EJB in this Servlet, and invoke methods on the EJB.
4) EntityManager injection will not work in Servlets. Instead use Persistence.createEntityManager to obtain EntityManager in the doGet() method and then invoke operations on it.

GOT THE ANSWER?

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

PHOTOGRAPH BY MARGOT HARTFORD
ART BY I-HUA CHEN

EDITORIAL

Editor in Chief
Justin Kestelyn

Senior Managing Editor
Caroline Kvitka

Community Editors
Cassandra Clark, Sonya Barry, Yolande Poirier

Java in Action Editor
Michelle Kovac

Technology Editors
Janice Heiss, Tori Wieldt

Contributing Writer
Kevin Farnham

Contributing Editors
Blair Campbell, Claire Breen, Karen Perkins

DESIGN

Senior Creative Director
Francisco G Delgadillo

Design Director
Richard Merchán

Contributing Designers
Jaime Ferrand, Paulina McFarland

Production Designer
Sheila Brennan

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

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