//MAY/JUNE 2014 /

Java™ magazine

By and for the Java community

TOOLS TO INNOVATE

Exploring IDEs, deployment tools, mobile, and the cloud

ORACLE.COM/JAVAMAGAZINE

ORACLE®

//table of contents /

05
7 OPEN SOURCE TOOLS FOR JAVA DEPLOYMENT
Step up your game on projects of any size.

Development Tools and Techniques

New theme icon. See how it works.

37
DEVELOPER TOOLS AND TRENDS
Oracle's Chris Tonas talks about the future of NetBeans IDE and more.

67
Enterprise
BUILD WITH NETBEANS IDE, DEPLOY TO ORACLE JAVA CLOUD SERVICE
Save time and effort deploying applications.

COMMUNITY

03
From the Editor

14
Java Nation
News, people, books, and events

22
JCP Executive Series
Java Tools for the Bottom Line
Werner Keil on the Java Money and Currency API

JAVA IN ACTION

28
Transforming Data into Information
Java applications from Pinkmatter Solutions turn seemingly unrelated data into useful information.

JAVA TECH

33
New to Java
Interactive Objects with BlueJ
Interactivity and visualization help beginners learn and form mental models.

42
Java Architect
Quick and Easy Conversion to Java SE 8 with NetBeans IDE 8
New tools in NetBeans 8 for leveraging the functional features of Java SE 8

46
Java Architect
Exploring Java 8 Profiles
What can Compact Profiles do for your application?

49
Java Architect
Processing Data with Java SE 8 Streams
Combine advanced operations of the Stream API to express rich data processing queries.

54
Process
A Sprint in the Life of a Scrum Master
Saskia Vermeer-Ooms gives a tour of a Scrum sprint.

59
Enterprise
HTML5 and JSF
Mix and match the two technologies to create sophisticated applications.

73
Enterprise
Mastering Binaries with Hudson, Maven, Git, Artifactory, and Bintray
A powerful tool chain that can be the backbone in a build/release and delivery pipeline

81
Rich Client
Mary Had a Little Lambda
Get familiar with lambdas and the Stream API through a simple game.

88
Rich Client
Leap Motion and JavaFX
Use 3-D hand movements to interact with JavaFX applications.

94
Fix This
Take our Java SE 8 Stream API code challenge!

COVER ART BY I-HUA CHEN

EDITORIAL

Editor in Chief
Caroline Kvitka

Community Editors
Cassandra Clark, Sonya Barry, Yolande Poirier

Java in Action Editor
Michelle Kovac

Technology Editor
Tori Wieldt

Contributing Writer
Kevin Farnham

Contributing Editors
Claire Breen, Blair Campbell, Karen Perkins

DESIGN

Senior Creative Director
Francisco G Delgadillo

Senior Design Director
Suemi Lam

Design Director
Richard Merchán

Contributing Designers
Jaime Ferrand, Arianna Pucherelli

Production Designers
Sheila Brennan, Kathy Cygnarowicz

PUBLISHING

Publisher
Jennifer Hamilton +1.650.506.3794

Associate Publisher and Audience Development Director
Karin Kinnear +1.650.506.1985

ADVERTISING SALES

President, Sprocket Media
Kyle Walkenhorst +1.323.340.8585

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

Eastern US and EMEA/APAC, Sprocket Media
Mark Makinney +1.805.709.4745

Advertising Sales Assistant
Cindy Elhaj +1.626.396.9400 x 201

Mailing-List Rentals
Contact your sales representative.

RESOURCES

Oracle Products
+1.800.367.8674 (US/Canada)

Oracle Services
+1.888.283.0591 (US)

Oracle Press Books
oraclepressbooks.com

ARTICLE SUBMISSION

If you are interested in submitting an article, please e-mail the editors.

SUBSCRIPTION INFORMATION

Subscriptions are complimentary for qualified individuals who complete the subscription form.

MAGAZINE CUSTOMER SERVICE

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

PRIVACY

Oracle Publishing allows sharing of its mailing list with selected third parties. If you prefer that your mailing address or e-mail address not be included in this program, contact Customer Service.

Copyright © 2014, Oracle and/or its affiliates. All Rights Reserved. No part of this publication may be reprinted or otherwise reproduced without permission from the editors. *JAVA MAGAZINE* IS PROVIDED ON AN "AS IS" BASIS. ORACLE EXPRESSLY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS OR IMPLIED. IN NO EVENT SHALL ORACLE BE LIABLE FOR ANY DAMAGES OF ANY KIND ARISING FROM YOUR USE OF OR RELIANCE ON ANY INFORMATION PROVIDED HEREIN. The information is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle. Oracle and Java are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

Java Magazine is published bimonthly with a free subscription price by Oracle, 500 Oracle Parkway, MS OPL-3C, Redwood City, CA 94065-1600.

Digital Publishing by GTxcel

//from the editor /

We are living in an age of innovation. Businesses across every industry are making technology-enabled innovation a priority. The next big thing is always just around the corner—maybe it's even an idea that's percolating in your brain.

In this issue, we explore some of the tools and techniques that can help you bring your ideas to fruition and make you more productive. Just look for our theme icon to find articles on this topic.

Development Tools and Techniques

In "Seven Open Source Tools for Java Deployment," Bruno Souza and Edson Yanaga present a set of tools that you can use now to drastically improve the deployment process on projects big or small—enabling you and your team to focus on building better and more-innovative software in a less stressful environment.

We explore the future of application development tools at Oracle in our interview with Oracle's Chris Tonas, who discusses plans for NetBeans IDE 9, Oracle's support for Eclipse, and key trends in the software development space. For more on NetBeans IDE, don't miss "Quick and Easy Conversion to Java SE 8 with NetBeans IDE 8" and "Build with NetBeans IDE, Deploy to Oracle Java Cloud Service."

We also give you insight into Scrum, an iterative and incremental agile process, with a tour of a development team's Scrum sprint. Find out if Scrum will work for your team. Other article topics include mastering binaries in Maven-based projects, creating sophisticated applications with HTML5 and JSF, and learning to program with BlueJ.

At the end of the day, tools don't make great code—you do. What tools are vital to your development process? How are you innovating today? Let us know.

Caroline Kvitka, Editor in Chief BIO

//send us your feedback /

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

PHOTOGRAPH BY BOB ADLER

Java magazine

Find the Most Qualified Java Professionals for your Company's Future

Introducing the *Java Magazine* Career Opportunities section – the ultimate technology recruitment resource.

Place your advertisement and gain immediate access to our audience of top IT professionals worldwide including: corporate and independent developers, IT managers, architects and product managers.

For more information or to place your recruitment ad or listing contact:
tom.cometa@oracle.com

Selecting an API for Working with Microsoft Office Files

Microsoft Office documents are everywhere. Learning the ins and outs of Microsoft Office file formats is time-consuming so instead many developers use a third-party API. How do you select one?

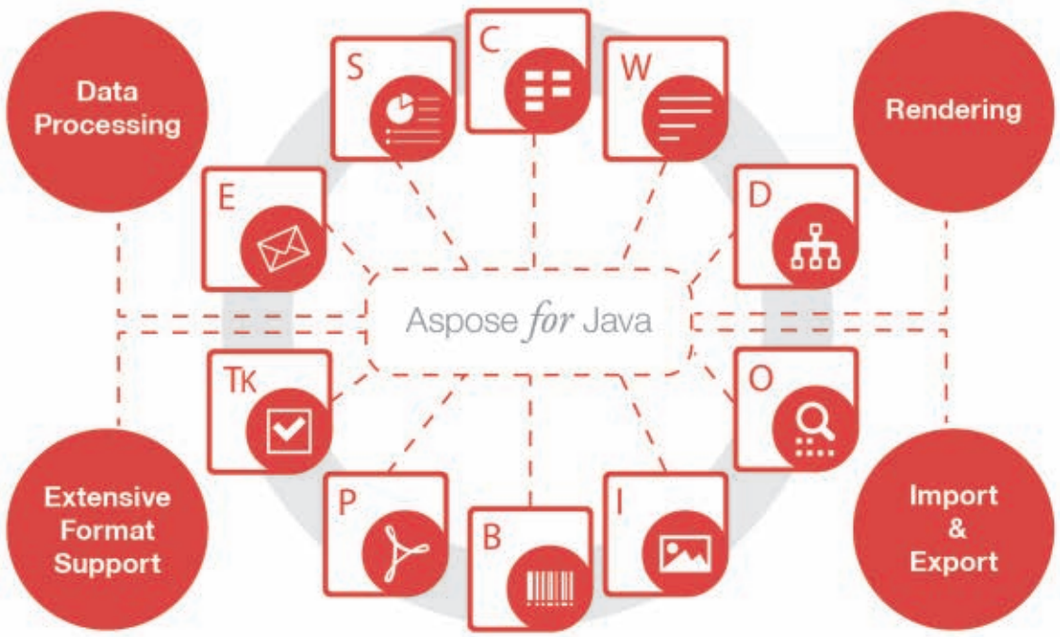
Open-Source

Apache POI is a collection of open-source APIs that offers a specific set of features for reading and writing Microsoft Office file formats. The benefit of open-source APIs is that they are free and open to customization. That's great if you have a lot of time and resources. The draw-back with open-source APIs is that they don't always have great support or documentation, and they support fewer features and variants. These drawback cost developers time, and reduce the dependability of their applications. Commercial APIs are a viable option.

Aspose for Java

Aspose for Java is a range of commercial Java APIs that help developers work with popular business file formats such as Microsoft Word documents, Microsoft Excel spreadsheets, Microsoft PowerPoint presentations, Adobe Acrobat PDF files, emails, images, barcodes and OCR.

Each API is designed to perform a wide range of document creation, manipulation and conversion tasks quickly and easily, saving time and letting developers get on with programming.



Aspose APIs support several advanced features, and files not handled by open-source alternatives. Go to www.aspose.com to find out more.

Feature-Rich

Before selecting a solution, decide which features you need. Aspose's APIs are feature-rich. Aspose.Cells, for example, is an API for working with Microsoft Excel spreadsheets that can create and modify spreadsheets, work with external data, create and update formulas, create, manipulate and export charts, work with Pivot Tables, apply formatting and page settings. If a user can do it in Microsoft Excel, a developer can do it with Aspose.Cells.

No open-source API has the same comprehensive feature support.

Efficient and Robust

All Aspose's APIs use a simple DOM and one API for working with a range of related formats. Aspose's Microsoft Office APIs, Aspose.Cells, Aspose.Words, Aspose.Slides, Aspose.Email, and Aspose.Tasks, are easy to work with, efficient, robust and independent of other libraries.

Dependable Support

Aspose for Java APIs are updated regularly and supported by free trials, code demos, extensive documentation, and support forums.

Running a trial, reading forums and documentation gives you confidence that the solution you chose can do exactly what you need it to.

ASPOSE
Your File Format APIs

Download a free trial of Aspose for Java today

7 OPEN SOURCE TOOLS FOR JAVA DEPLOYMENT

Step up your game on projects of any size. **BY BRUNO SOUZA AND EDSON YANAGA**

Development Tools and Techniques

Continuous deployment is a set of automation practices that reduces lead time and improves the reliability, quality, and overhead of software releases. Implementing continuous deployment requires some work, but it has a very positive impact on a project. By establishing a sound deployment pipeline and keeping all of your environments—from development to test to production—as similar as possible, you can drastically reduce risks in the development process and make innovation, experimentation, and sustained productivity easier to achieve.

Developers are usually aware of the basic building blocks for deployment: code repositories, such as Git and

ART BY I-HUA CHEN

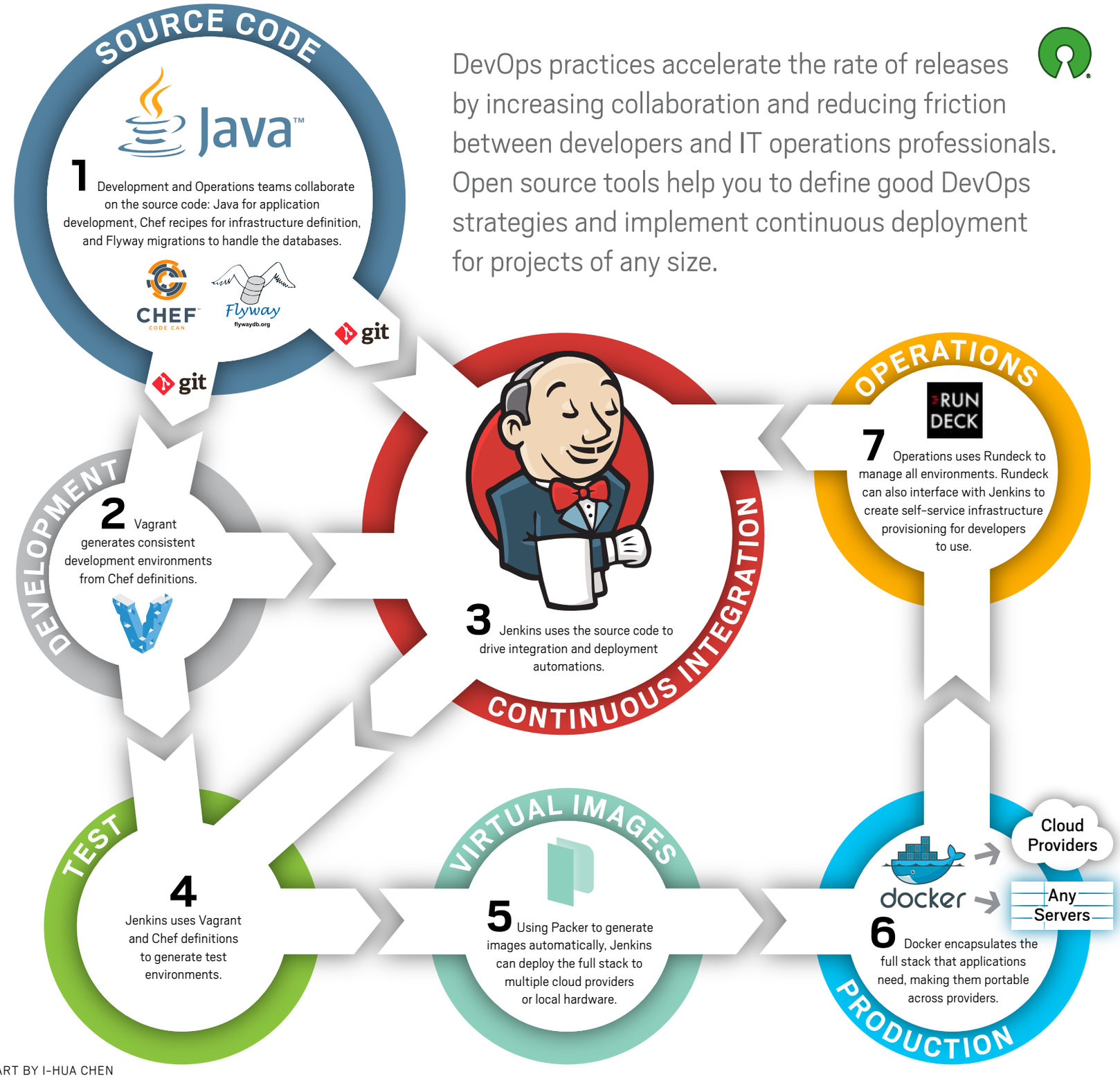
Subversion; and build tools, including Ant, Maven, and Gradle. But what other tools can help you step up?

In this article, we present seven open source tools that you can use right now to improve the deployment process on projects big or small. These tools are among the best and most-used tools in their areas; they attract developers who have created a large body of knowledge, plugins, and connectors that can be used in a wide range of situations and integrated with other tools and processes.

But more importantly, these tools can dramatically improve your deployments. They can empower your team to build better and more-innovative software in a less stressful environment.

Open Source Tools for Continuous Deployment

DevOps practices accelerate the rate of releases by increasing collaboration and reducing friction between developers and IT operations professionals. Open source tools help you to define good DevOps strategies and implement continuous deployment for projects of any size.



ART BY I-HUA CHEN

1 JENKINS

Start with Continuous Integration

Released early in its life as the Hudson continuous integration (CI) server, Jenkins is the most active and most used CI server today. (For more on Hudson, check out "Mastering Binaries with Hudson, Maven, Git, Artifactory, and Bintray," by Michael Hüttermann, in this issue.)

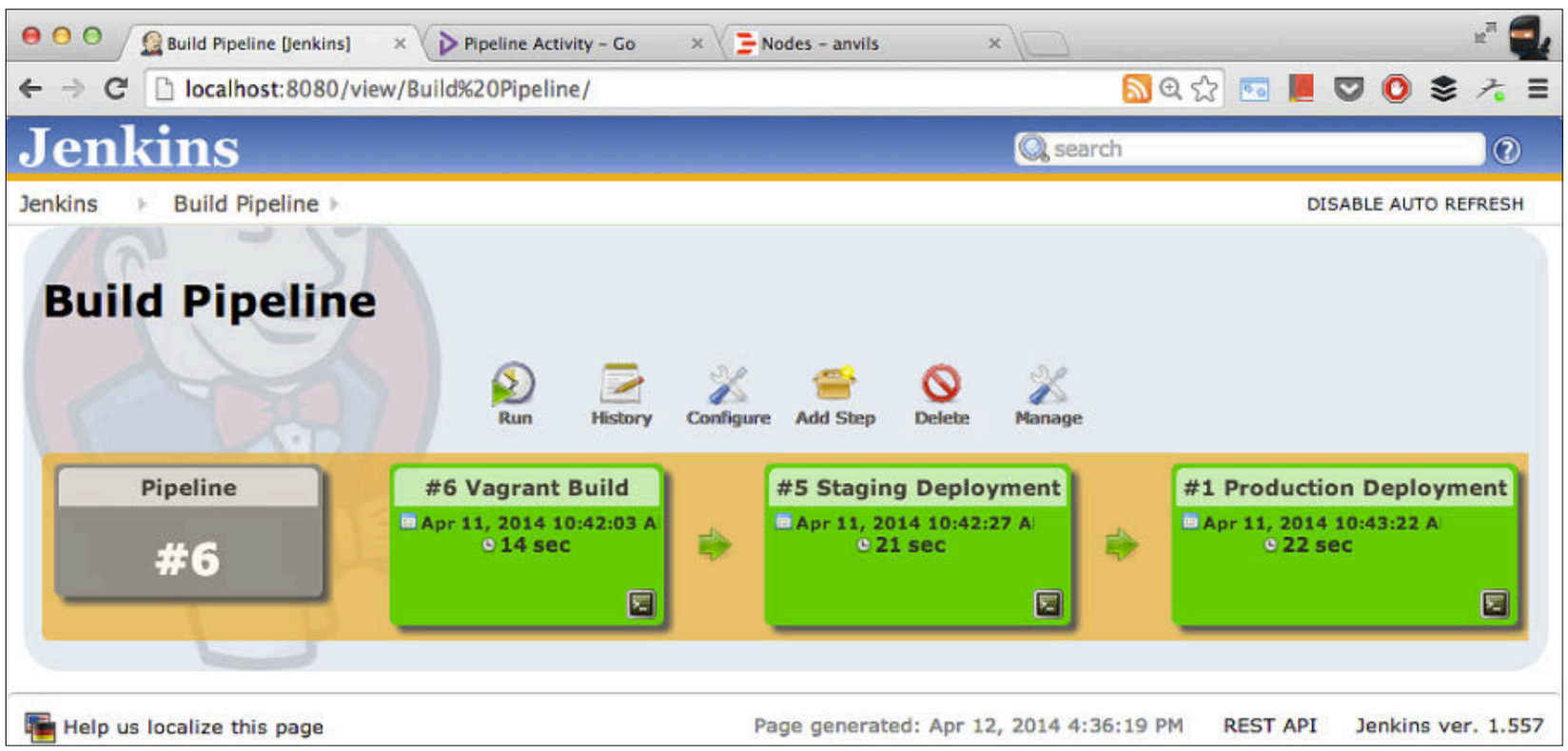
Jenkins is the cornerstone of automation in your project, no matter what language your project is written in. But with its origins—and popularity—in the Java community, Jenkins is particularly

well suited for Java projects from desktop installers to remotely deployed web application archive (WAR) files to application servers in the cloud.

Because of its extensive plugins community, Jenkins can connect to diverse systems. With its flexible and powerful job definition mechanism, you can use any kind of build tools to automate every part of the development process. This functionality creates a system that can grab information from multiple sources, run the build steps to create nearly any type of application you can think of, and connect to other target systems to deploy the application to whatever infrastructure is needed.

To build a good deployment process, you need to grab a few of those plugins. Here are some that can't be left out of any deployment environment:

- **Build Pipeline Plugin.** The Jenkins delivery approach involves the chaining of related jobs through build triggers. This flexible functionality lets you create sophisticated pipelines, but it can be difficult to see the pipeline's "whole picture," the relationships between jobs, and where in the pipeline the build is at a given moment. The Build Pipeline Plugin solves that problem by providing a nice overview of the build pipeline, so you can easily follow the build as it happens and even decide when things should be automatic or require a manual trigger.
- **Parameterized Trigger Plugin.** In a build pipeline, developers use the artifact that is generated as output from one job as the input for the next job in the pipeline. The Parameterized Trigger Plugin informs the next job which build result it must use to keep the pipeline moving.
- **Copy Artifact Plugin.** A complement of the Parameterized Trigger Plugin, the Copy Artifact Plugin uses the parameters received from the previous job to fetch that job's resulting artifacts and uses them as the starting point of the next job.



Power and flexibility in Jenkins: The Build Pipeline Plugin makes it easy to chain Jenkins jobs to organize a deployment pipeline.

2 CHEF

Turn Your Infrastructure into Source Code

Chef is a provisioning automation framework that simplifies the configuration of your development or production environment—whether on-premises or cloud-based. It is a Ruby-based tool that does wonders to deploy any infrastructure needed for your Java application.

By using Chef, you can define a project's infrastructure with Ruby scripts that are versioned within the project's source code repository. Chef scripts are called *recipes*, and they are bundled in *cookbooks*. With scripts written in Ruby, you have a well-known, generic scripting language to automate infrastructure activities.

Any time you need to evolve the

infrastructure, you can evolve the scripts and rebuild the full environment with the new definitions. By doing so, Chef replaces lots of documented (and undocumented) manual installations, configurations, and ad hoc decisions, and it creates clean, versioned, automated steps to generate the infrastructure. This powerful concept allows the infrastructure definition to evolve with the application and promotes team interaction. Having a common tool, a common repository, and a clear deployment process goes a long way to integrate the development and operations teams.

Chef handles all kinds of infrastructure components: from your operating system and the software that needs to be installed to any configuration you need to apply, such as users and IP addresses. But Chef can go much further if needed, handling firewalls, network devices, multiple servers, cloud environments, and other components that form your infrastructure. Besides building your application environment, Chef includes a client/server architecture that lets you centrally manage patches and software upgrades on multiple servers.

The easiest way to start using Chef is with its *chef-solo* client version, which simplifies environment provisioning so you can build the necessary infrastructure for your project with a single command. After you have your cookbooks, you can use knife, Chef's command-line interface (CLI), to prepare a server by installing chef-solo in it, as shown in **Listing 1**. And you can automatically provision the server from the cookbooks, as shown in **Listing 2**.

LOSE THE STRESS

These tools can dramatically improve your deployments. They can empower your team to build better and more-innovative software in a less stressful environment.

LISTING 1 LISTING 2 / LISTING 3

```
$ knife solo prepare host_username@host
```

Download all listings in this issue as text

3 VAGRANT

Reproduce Development Environments

Of all the tools mentioned in this article, Vagrant is probably the least concerned with deployment per se. Vagrant is focused on the developer, but it bridges development and production, helping to reduce inherent risk by minimizing discrepancies between the different environments of the deployment process.

A common problem developers have is constructing a local development environment in which to run the application they're building. Target systems are becoming more sophisticated and complex, with multiple web and application servers, databases, library dependencies, queues, service integration frameworks, caches, and load balancers, among other elements. This situation complicates the possibility of having development and test environments that closely resemble the final production environment, making it difficult to create a reliable build-test-deploy pipeline.

Vagrant generates virtual development environments from a text file definition called the *Vagrantfile*. You can have this definition checked in with the source code of your project, and every developer can then use Vagrant by running a single command to provision a fully functional development environment identical to everyone else's. Locally, this environment is created as a virtual machine running within Oracle VM VirtualBox, Oracle's open source virtualization product.

After you have a Vagrantfile configured, you run a single command:

```
$ vagrant up
```

Here's how Vagrant meets deployment: your CI server can use the Vagrantfile to build virtual machines for your test and quality assurance (QA) environments. Then later—especially if Vagrant is integrated with Chef, Packer, or Docker—the Vagrantfile forms the basis for your continuous deployment

process to build your final production environment.

At first look, Vagrant might seem similar to Chef, but in fact, they work great together. Chef helps you provision your infrastructure and build your systems from development to production. Vagrant, however, helps you create provisioned virtual machines; it can use Chef to do the actual provisioning. Even better, Vagrant works amazingly well with Chef's chef-solo client to build development and test environments. This functionality allows Chef to define your infrastructure, and you can run Vagrant to have a development environment that's similar to the target production system. All environments—development, test, QA, and production—can evolve together from the same Chef description, which is versioned, tagged, and kept in your code repository.

4 PACKER

Generate Images for Multiple Environments

In a "normal deployment," you *provision*, or prepare, your systems for operation. Starting from a base system, you install all the needed software until you have a fully functional environment. Tools such as Chef help you automate the provisioning process, making it repeatable and easy to evolve.

However, virtualization and cloud computing changed normal deployment. Today, your "base system," which is usually encapsulated into a virtual image, can be prepared with everything you need, all the way to the application level and even the data. What we used to call provisioning could take some time, but now it's simply starting a virtual machine and can be accomplished in seconds with everything ready to go.

Building new virtual images is a painful process. Because of that, many teams keep large portions of their solution outside the image. Then, after the virtual image is instantiated, they run extra processes to update the application from the repository, download

It's all in the image: Packer bundles the environment and the application, producing a ready-to-run virtual image appliance.

```
jack-javaone — vagrant@rundeck:~ — packer-provision — 102×31
vagrant@precise64: ~   vagrant@rundeck:~   java   vagrant@precise64: ~
$packer build javaone.json
amazon-ebs output will be in this color.

==> amazon-ebs: Creating temporary keypair: packer 534996df-6602-7fa9-ce04-28749d054a38
==> amazon-ebs: Creating temporary security group for this instance...
==> amazon-ebs: Authorizing SSH access on the temporary security group...
==> amazon-ebs: Launching a source AWS instance...
    amazon-ebs: Instance ID: i-f68fbda6
==> amazon-ebs: Waiting for instance (i-f68fbda6) to become ready...
==> amazon-ebs: Waiting for SSH to become available...
==> amazon-ebs: Connected to SSH!
==> amazon-ebs: Provisioning with chef-solo
    amazon-ebs: Installing Chef...
    amazon-ebs: % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
    amazon-ebs: Dload  Upload   Total   Spent    Left  Speed
    amazon-ebs: 100 14401  100 14401    0     0  56452      0 --:--:-- --:--:-- --:--:-- 76195
    amazon-ebs: Downloading Chef  for ubuntu...
    amazon-ebs: downloading https://www.opscode.com/chef/metadata?v=&prerelease=false&p=ubuntu&pv=12.0
4&m=x86_64
    amazon-ebs: to file /tmp/install.sh.892/metadata.txt
    amazon-ebs: trying wget...
    amazon-ebs: url     https://opscode-omnibus-packages.s3.amazonaws.com/ubuntu/12.04/x86_64/chef_11.
12.2-1_amd64.deb
    amazon-ebs: md5     cedd8a2df60a706e51f58adf8441971b
    amazon-ebs: sha256  af53e7ef602be6228dcbf68298e2613d3f37eb061975992abc6cd2d318e4a0c0
    amazon-ebs: downloaded metadata file looks valid...
    amazon-ebs: downloading https://opscode-omnibus-packages.s3.amazonaws.com/ubuntu/12.04/x86_64/chef
_11.12.2-1_amd64.deb
    amazon-ebs: to file /tmp/install.sh.892/chef_11.12.2-1_amd64.deb
    amazon-ebs: trying wget...
    amazon-ebs: Comparing checksum with sha256sum...
```

data, and adjust configurations. That way, the base image stays reasonably stable over the course of many deployments. This process is good, but it complicates the evolution of the environment. It can create weird failures when the application is deployed in an image that doesn't have the latest infrastructure improvements to support it.

This is where Packer comes in. Packer is a powerful command-line utility that automatically generates images for different providers (Oracle VM VirtualBox, VMware, AWS, DigitalOcean, Docker, and so on). It can run at the end of your CI process, and it reuses your Chef provisioning. The result of Packer's execution—which can take some time, depending on your provisioning needs—is a fully functional, everything-integrated, configured, and running virtual machine image that's ready to start.

Because Packer generates the image automatically, you can run it at every release build and include in the image the latest version of your application, already deployed in the configured application server, with the data that needs to be there. You just need to start an instance and your system is up and running in a few seconds. It's ready to increase the pool in your load balancer, replace a failing server, or take over the job from the server that was running the previous version.

BIG IMPACT
Implementing continuous deployment requires some work, but it has a very positive impact on a project.

All you need is a Packer template with your image definition, and then you run the following command:

```
$ packer build template.json
```

Try Self-Sufficient and Portable Lightweight Containers

Creating ready-to-boot virtual machine images is a sophisticated way to use cloud computing so you can scale your application. But dealing with full stacks, complete operating systems, application servers, and the rest might be overkill. In many cases, all you want is to add one more application to your stack.

Platform-as-a-service (PaaS) solutions have been well received by developers, because they can add their application to a predefined stack with little overhead to worry about. Simply push the code, and it runs. But this approach might be too simple. It requires that you run your application in a shared environment inside a cloud provider, with whatever stack is provided, and sometimes with less flexibility than you'd like.

This is where Docker comes in. Built on top of Linux Containers (LXC), Docker creates portable, self-sufficient containers that include your application. Docker can be run locally on your computer or scaled to cloud deployments. By encapsulating everything your application needs in the container abstraction, Docker makes deployment easy. At the same time, because Docker deals with containers, your application doesn't need to carry all the baggage of a full operating system installation. Docker containers can be extremely lightweight, so you can run many containers—even on small systems.

You can configure a Docker image to include everything you need, as a PaaS provider would. And then, each container can host your application, with only what needs to be changed from application to application. Containers are easy to start, stop, manage, and evolve. They are also isolated, and applications can have their own version of libraries.

Docker can be used both to create a PaaS-like environment and to make your CI process easier by using containers to manage test and QA environments with the same container definition that's put into production later. Docker also integrates well with Chef and Packer, and you can use those tools to generate a container from your application automatically. This same container can be used locally for builds and tests; it can be run by your CI server for integration and deployment; and it can scale to virtual machines, local servers, and private or public clouds to run your production.

After you have a *Dockerfile* config-

ured, you can easily start new containers by running the following command:

```
$ docker run image/name command
```

For example, Quinten Krijger has an Apache Tomcat image that you can download by running the following command:

```
$ docker pull quintenk/tomcat7
```

And then you can use the following command to run the image as a daemon:

```
$ docker run -d quintenk/tomcat7
```

```
3   import com.googlecode.flyway.core.api.migration.jdbc.JdbcMigration;
4
5   import java.math.BigDecimal;
6   import java.sql.Connection;
7   import java.sql.ResultSet;
8   import java.sql.Statement;
9
10  public class V1__Complex_Migration implements JdbcMigration {
11
12      private static final String UPDATE_10_PERCENT =
13              "UPDATE CUSTOMER SET CREDIT_SCORE = CREDIT_SCORE * 1.1 WHERE ID = %d";
14
15      private static final String UPDATE_ANOTHER_1_PERCENT =
16              "UPDATE CUSTOMER SET CREDIT_SCORE = CREDIT_SCORE * 1.01 WHERE ID = %d";
17
18      @Override
19      public void migrate(Connection connection) throws Exception {
20          try (Statement statement = connection.createStatement()) {
21              try (ResultSet resultSet = statement.executeQuery("SELECT ID, CREDIT_SCORE FROM CUSTOMER")) {
22                  while (resultSet.next()) {
23                      long id = resultSet.getLong("ID");
24                      BigDecimal credit_score = resultSet.getBigDecimal("CREDIT_SCORE");
25                      if (credit_score.compareTo(new BigDecimal("1000.00")) > 0) {
26                          statement.executeUpdate(
27                                  String.format(UPDATE_10_PERCENT, id));
28                      }
29                      else if (credit_score.compareTo(new BigDecimal("10000.00")) > 0) {
30                          statement.executeUpdate(
31                                  String.format(UPDATE_ANOTHER_1_PERCENT, id));
32                      }
```

6 FLYWAY

Database Migrations that Follow Your Applications

We can't talk about deployment without mentioning databases. Developers understand how to upgrade an application, maintain compatibility with older versions of APIs, and deprecate functionality. But what about the database? The hardest part of an automated deployment can be underdocumented SQL scripts that need to be run "just before" production, differences between the code and the schema, and evolution and rollback situations.

Flyway shines in these situations. Developed in Java and focused on the needs of Java developers, Flyway is a database migration framework that can migrate from any version of your database schema to the latest version. Flyway keeps the database definition of your application safe inside your version control system and turns your database into a clear, precise, and versioned set of instructions that can be re-created or migrated every time that is needed.

Flyway has a CLI utility and also integrates with build tools such as Maven and Ant. It can run as part of your CI process to upgrade or create a database before running tests or going into QA. Flyway can run SQL scripts or—if you have sophisticated database needs—specific Java code to handle migrations through the JDBC API. Through its API, you can also manage migrations from inside your application to implement database management functionality. If you are distributing an application, Flyway can handle the database creation or migration on the first run.

By turning your database schemas into code in your repository, Flyway provides visibility into the database and helps the whole team participate in its evolution. This integration pushes everyone to work together toward continuous deployment.

Powerful database migrations: Flyway can leverage the use of the JDBC API to handle schema migrations.

With Flyway configured, you can create or migrate your databases using this CLI command:

```
$ flyway migrate
```

7 RUNDECK

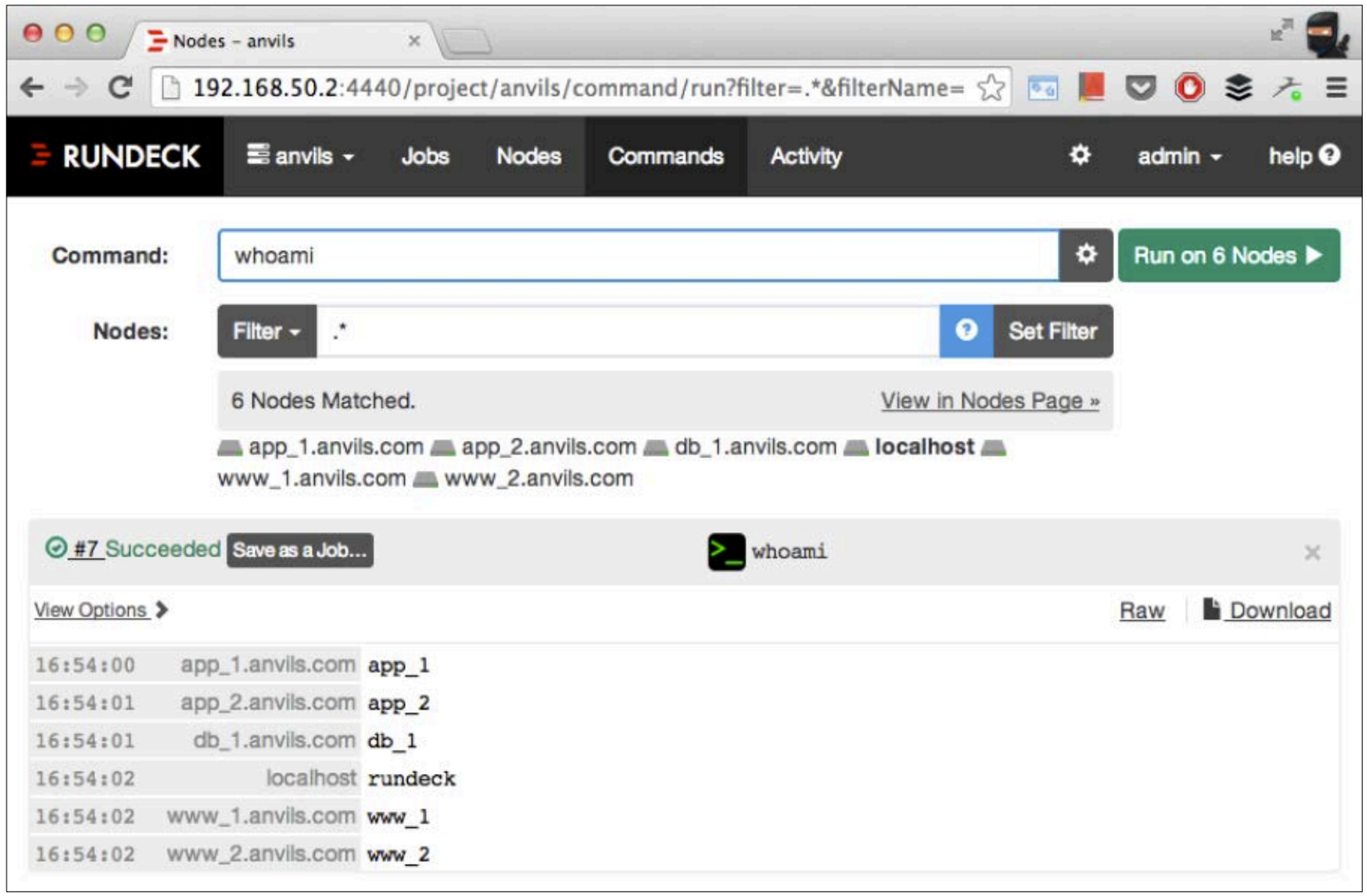
Support the *Ops* in *DevOps*

That brings us to *DevOps*, a term that means communication, collaboration, and integration between developers and IT operations professionals. As developers, we know how good tools promote collaboration. When we save infrastructure definitions in our repositories or we automate infrastructure creation in a way that benefits both developers and IT operations professionals, we promote DevOps strategies in our projects. Each of these seven tools helps the collaboration between development and operations.

Collaboration is important so we can deliver the software products and services that customers expect. But each group has a different view of what software and services are. While developers think about code, bugs, libraries, and dependencies, IT operations professionals consider servers, nodes, security, and auditing. Developers worry about functionality; operations professionals worry about availability. Developers think that automation means building the source code and generating artifacts for deployment. Operations professionals might call automation the discovery of new nodes and the dispatching of commands to multiple servers. These different views focus on the same servers and the same services, and they're both responsible for getting working software to the end user.

Rundeck, a Java-based web application, helps the integration and collaboration of these two worldviews by functioning as the operations console for your environment. Rundeck knows about the details of the environment, such as the nodes you have and the services you're running. It can execute actions in the environment, by either running predefined jobs or executing ad hoc commands on one or more nodes.

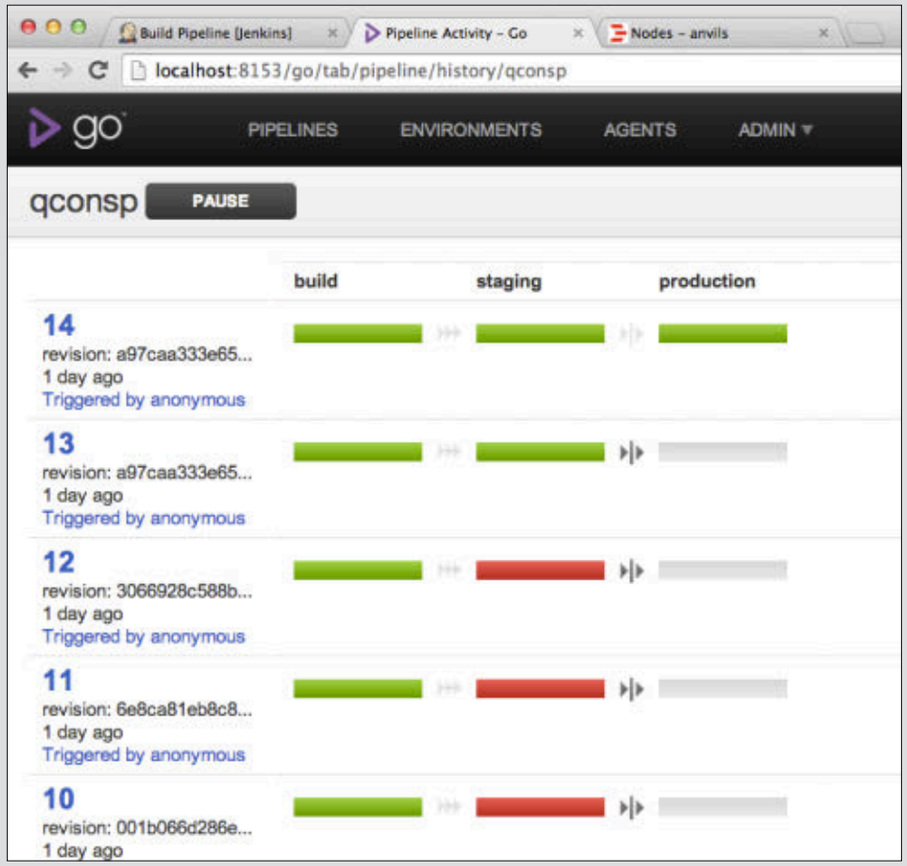
Rundeck integrates beautifully with the other tools in this set, and it retrieves detailed environment information from Chef servers or collects node information directly from your cloud provider—so you have an updated view, even in a



Managing multiple nodes: Rundeck runs commands in several nodes, and aggregates the results.

Go: One More Tool

ThoughtWorks has just released Go, a continuous integration and release management server, as an open source, Apache-licensed project. Go is a Java-based server and has a role similar to Jenkins, but the two have slightly different concepts. Constructed around the full build pipeline rather than around independent jobs, Go focuses on the code-build-test-deploy process. This structure makes continuous deployment a main focus in your project and helps the team think about it from the start. It's still too early to say whether developers will accept Go and what place it will claim in this market. However, it's a great concept and a fully functional tool that can be easily integrated with the seven other tools.



Ready-to-run deployment: Go's built-in pipeline clearly shows successful, failed, and manually triggered builds.

dynamic cloud environment. Rundeck then executes actions on a single node or on hundreds of nodes. It does that either directly through Secure Shell (SSH) or by integrating with Chef or other tools, and by defining filters to decide what to run and where. The results and outputs are then aggregated, helping make sense of what is happening.

A secure web dashboard provides access control to Rundeck jobs. Jobs are also turned into API calls that can be called from scripts through a CLI or directly from Jenkins jobs through a plug-in that further simplifies the integration. This allows operations professionals to define actions that developers can run, creating self-service, on-demand IT services. Thus, IT procedures essentially are automated, encapsulated, and made easily available for the build process to call into. Operations professionals are part of the process and define what can be done. But they don't become a bottleneck, because the services they define can be run by developers when needed.

The Rundeck CLI is a powerful management tool. If you want to copy and run a script to all your UNIX machines, you can simply run the command shown in **Listing 3**.

CONCLUSION

The tools described in this article are all open source tools, and you can implement them today in your project. They can be used to deploy all kinds of applications, and are particularly well suited to deploying Java applications and services.

Try these tools and experience how they help you to move toward a full deployment pipeline for your Java projects. `</article>`

MORE ON TOPIC:

Bruno Souza is a Java developer and open source evangelist at Summa Technologies and a cloud expert at ToolsCloud. He is the founder and coordinator of SouJava—one of the world's largest Java user groups—founder of the Worldwide Java User Groups Community, and director of the Open Source Initiative (OSI).

Edson Yanaga is a technical lead at Produtec Informática and a principal consultant at Ínsula Tecnologia. An open source user, advocate, and developer, his expertise encompasses Java, application lifecycle management, cloud computing, DevOps, and software craftsmanship. He is a frequent speaker at international conferences.

LEARN MORE

- Source code demo
- Bruno Souza's blog
- Edson Yanaga's blog

Clockwise from top: Mike Milinkovich ponders the future; Thomas Schindl gives a JavaFX 8 update; Stuart Marks (left) and Georges Saab kick off Java 8 Day.

JAVA 8, ECLIPSE, AND THE FUTURE

PHOTOGRAPHS BY BOB ADLER/GETTY IMAGES

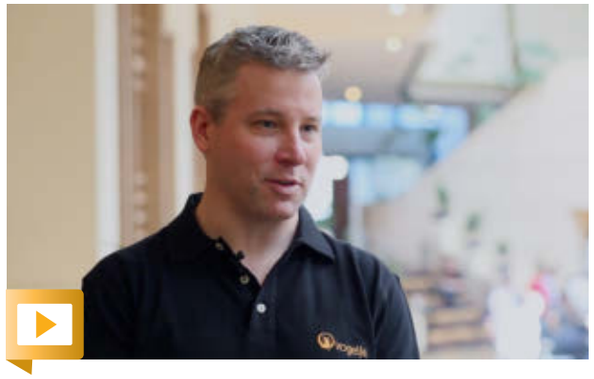
EclipseCon 2014, held March 17–20 in San Francisco, California, boasted 14 tutorials, 120 sessions, and four theme days—including Java 8 Day on the second day of the conference. Java 8 launched a few hours into the day, which added an air of excitement and talk of taking down the hotel's Wi-Fi with all of the downloads. The fact that all of the major IDEs provided Java 8 support that day was a sign that sometimes the planets just align.

Day two of the conference kicked off with the keynote "Eclipse: The Next Ten Years" by **Mike Milinkovich**, executive director of the Eclipse Foundation. After discussing the history and success of Eclipse, Milinkovich provided the caveat that he does not have the power to dictate what projects will occur in the Eclipse community. "I think of my title as chief Eclipse cheerleader," he said. "It's a very Darwinian, bottom-up process. What survives is what works," he explained.

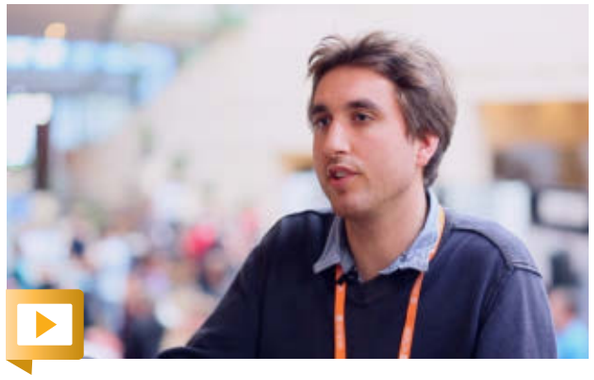
Milinkovich talked about three trends that will have an impact on developers and IDEs, and they apply to Java and the open source community as well.

Trend #1: Software Is Eating the World. With a nod to **Marc Andreessen**, Milinkovich said that software is becoming ever more important and has an effect on everything, including how companies are valued. He gave the example of the Airbus aircraft: the amount of software code used onboard on this class of aircraft

Thomas Schindl explains what's new in JavaFX 8.

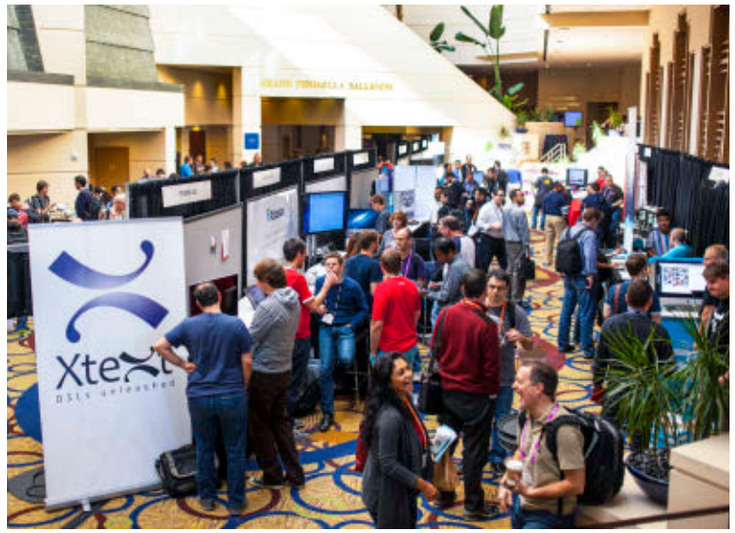


Denis Roy, IT director at the Eclipse Foundation, discusses the conference.



Tristan Faure of Atos talks about why he attends EclipseCon.

PHOTOGRAPHS AND VIDEOS BY BOB ADLER/GETTY IMAGES



grew four times larger in three years. So should Airbus think of itself as an airplane company or a software company? Not only is the codebase huge, but the lifespan of applications can span generations. "Would you program differently if your granddaughter will have to maintain your code?" Milinkovich asked with a smile.

Trend #2: The Internet of Things. Whether the Internet of Things (IoT) is a US$14 trillion market or whether that's hype, IoT is big and will continue to grow, he said, and most importantly, we need open IoT. Milinkovich said that Eclipse has 14 projects in the IoT space, with more on the way. There was a lot of interest in IoT throughout the conference.

Trend #3: The Cloud. Evans Data predicts that by 2019, 65 percent of developers will primarily develop for cloud, he said. Does that mean all the functionality from current desktop IDEs should be moved to the cloud? Milinkovich introduced a demo of Project Flux, which showed how to connect an Eclipse project to the cloud. EclipseCon offered multiple sessions on developing in the cloud.

JAVA 8 DAY

Java 8 Day at EclipseCon kicked off with the standing-room-only session "New Features in Java SE 8," with Oracle's **Georges Saab**. Java 8 supports the basic definition of Java, he said. It is simple, stable, fast, scalable, and easy to read. Saab also gave his "top 8" reasons to use JDK 8: lambda expressions and the Stream API, the new Date and Time API, Compact Profiles, Nashorn, Java Flight Recorder and Java Mission Control, the end of permanent generation, updated standards, and the fact that it was developed in OpenJDK. Saab encouraged everyone to move to Java 8 as soon as possible, join a Java user group (JUG), and check out OpenJDK and Adopt OpenJDK.

Java 8 Day continued with another full session on lambda expressions with **Alex Buckley**, Specification Lead for the Java language and Java Virtual Machine (JVM). He described lambda expressions as "perhaps the biggest upgrade ever to the Java programming model" but noted that Java 8 is much more than lambdas. He mentioned the new Stream API and said, "The jump from collections to streams is larger than the jump from anonymous classes to lambda expressions."

Other sessions at Java 8 Day included **Thomas Schindl** on what's new in JavaFX 8, and Oracle's **Hinkmond Wong** on Java SE 8 Compact Profiles for embedded development. It was a great day for Java 8.

FEATURED JAVA USER GROUP

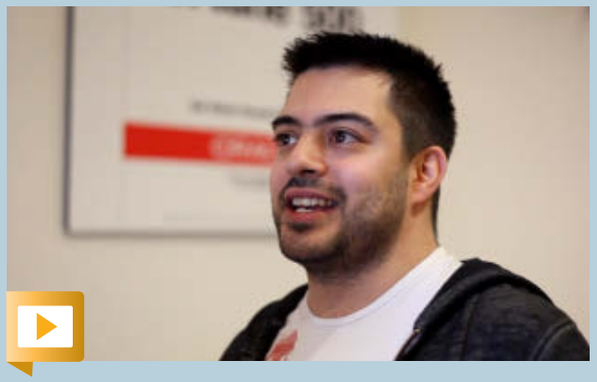
VIRTUAL JUG (vJUG)

The Virtual Java User Group (vJUG) was founded in October 2013 by Simon Maple. vJUG is an online user group aimed at Java developers who don't live near an active JUG, or who want more technical content. Maple notes, "Our goals are simple: to provide great technical content from the best speakers to a global community of developers, regardless of location, time zone, or country."

Although vJUG has only been meeting since November 2013, it already has more than 1,000 members from more than 50 different countries. To date, vJUG has been meeting two or three times each month. "We hold our meetings online using Google Hangouts On Air, YouTube, and good old IRC [Internet Relay Chat]," says Maple. "Google Hangouts On Air allow 10 speaking participants only, but we connect it with our vJUG YouTube channel to allow unlimited viewers, live. This provides access to our content without the need for people to download or install anything on client machines. We also use IRC as the mechanism for participants to talk about what's being presented live, as well as ask questions to others in the chat."

vJUG organizers are also looking into the possibility of doing online conferences and hack sessions. Maple also hopes to get vJUG involved in the Adopt-a-JSR and Adopt OpenJDK programs in the coming year.

Maple describes Adopt-a-JSR and Adopt OpenJDK as "two great programs aimed at reducing the barriers to obtaining the average developer's input and feedback as early as possible, to benefit and direct the future Java releases, codebase, and standards. I'm also a member of the London Java Community, so I have seen both these programs working well. It's our job as JUG leaders to bring these resources to our members, the individuals who can participate, which is an important step."



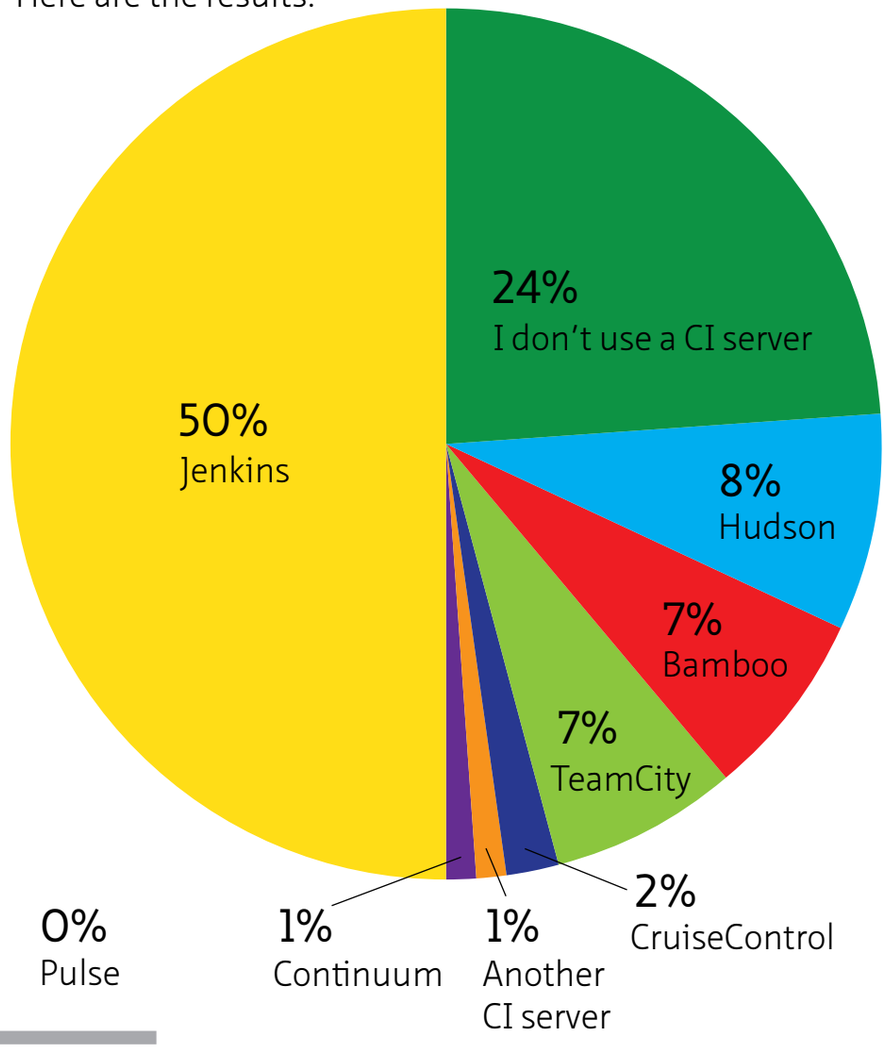
Simon Maple explains the goals of the Virtual Java User Group.

VIDEO BY BOB ADLER/GETTY IMAGES

JAVA.NET POLL

Jenkins Is the Preferred Continuous Integration Server

In a recent Java.net poll, Java and Java Virtual Machine (JVM) developers who utilize a continuous integration (CI) server indicated an overall preference for Jenkins. The poll, which asked, "Which Continuous Integration (CI) server do you prefer?" was answered by 260 developers. Here are the results:



Robert Savage, creator of the Pi4J project

PI4J PROJECT

Robert Savage grew up tinkering with and programming Commodore and Tandy computers. He has been programming for more than 25 years, working in the automation industry for nearly 15 years. A software architect for AMX, Savage says, "It was natural for me to relate to the motivations that inspired the Raspberry Pi founders." This led Savage to create the Pi4J project: "I believed the Raspberry Pi would play an instrumental role in helping usher in the Internet of Things [IoT]. Also, as a longtime consumer of numerous open source platforms, frameworks, and tools, I felt called to give back."

Before Savage created the Pi4J project (@pi4j), "there were a few other community projects that provided Java programmers access to the GPIO [general-purpose input/output]," he says. "However, these projects provided direct-access APIs where the programmer was forced to code in a very procedural style. Pi4J's mission is to create an object-oriented and event-driven API to make working with the Raspberry Pi's I/O capabilities more Java-friendly."

Savage experimented with Pi4J prototypes in the summer of 2012. The first Pi4J commit to GitHub happened in September 2012. "The project has grown steadily since its inception," he says. "However, in the past six months there seems to be a sharp incline in the number of e-mails and forum thread activity."

At present, there are three principal Pi4J developers, and a number of community members contribute fixes and gvf enhancements. The current total download count exceeds 15,000.

Pi4J started out providing simple Java-based access to GPIO, but current versions include GPIO, RS232, SPI, and I^2C capabilities, as well as support for GPIO expanders. The project is moving steadily toward an official release of Pi4J Version 1.0. For this release, "focus is shifting from raw I/O access to creating higher-level component and device interfaces and abstractions," Savage says. "Rather than programmers controlling and interacting with a GPIO pin, they can instead create their own reusable components. The intent is to abstract away the low-level I/O and provide interfaces for working with devices and real-world things. It's a work in progress, but you can see some examples in the 'develop' branch in GitHub."

Savage expects the IoT to bring "an explosion of interesting new devices that will make our lives easier through connectivity," he says. "I imagine intelligence added to ordinary things that ultimately leads to collaboration between devices to coordinate a fully automated lifestyle. With funding platforms such as Kickstarter and Indiegogo, I see a wealth of opportunity for independents to build and create."

You can follow Savage's endeavors on his blog or on Twitter (@savageautomate). His JavaOne 2013 session, "Let's Get Physical: I/O Programming with Java on the Raspberry Pi with Pi4J," is also available on Parleys.

NightHacking Java 8 Tour

From March 11–April 29, Oracle's Java Evangelist extraordinaire Stephen Chin made his way through Europe and India on the NightHacking Java 8 Tour.

Chin rented a motorcycle in Munich, Germany, and loaded it up with video and demo equipment. He motored from Poland to Spain, stopping at four conferences and visiting more than 20 Java user groups.

Along the way, Chin live-streamed events, hack sessions, and random encounters and kept everyone updated on Twitter.

Chin wrapped things up with a trip to India to attend the Great Indian Developer Summit in Bangalore, India.

JAVA CHAMPION PROFILE

ZORAN SEVARAC

Zoran Sevarac (right) strums a tune with fellow Java Champion Frank Greco.

***Zoran Sevarac** is an assistant professor at University of Belgrade, Serbia. He was selected to be a Java Champion in November 2013.*

Java Magazine: Where did you grow up?

Sevarac: In Belgrade, Serbia.

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

Sevarac: At the age of 12, when I figured out how to color the TV screen, and, even more fascinating, how to make it randomly change colors. A few years later, I made a space shooter game in Pascal.

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

Sevarac: ZX Spectrum, Basic.

Java Magazine: What was your first professional programming job?

Sevarac: I worked as a web developer in an e-commerce startup—Net Link Solutions—in Serbia with a few friends. Now it has seven employees.

Java Magazine: What is your current job?

Sevarac: I work at the University of Belgrade as an assistant professor in software engineering. I teach courses about software development in Java, and also work as a researcher in the Laboratory for Artificial Intelligence.

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

Sevarac: Programming, gaming, science fiction, and music.

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

Sevarac: The opportunity to meet and work with interesting people and travel around the world; also getting involved in the community.

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

Sevarac: I became a Java Champion recently, and it hasn't changed much yet, but I think it will in the future. I started thinking, "OK, I'm a Java Champion; now what do I do?" And it came to me: the community recognition that comes from being a Java Champion could help me carry out initiatives related to the open source project I'm involved in.

Java Magazine: What do you have on your mind?

Sevarac: The main project I'm working on is Neuroph, which is a popular Java open source neural network framework. I would like to build a stronger community around it, and closely collaborate with other related Java AI projects, maybe building a common API for some typical problems/tasks. Ideally, I'd talk them into moving some parts to the NetBeans platform, so we can easily integrate and exchange components as plugins. That would be beneficial for all AI projects, the whole Java AI community, and also the projects that rely on them.

Zoran Sevarac also participates in the Good Old AI Research Network. Read his Extreme Mind Refactoring blog and follow him on Twitter (@neuroph).

JAVA EE 8 COMMUNITY SURVEY

In the third and final part of the Java EE 8 Community Survey, each of nearly 4,500 developers allocated 100 points across 13 potential Java EE 8 enhancements. That's nearly a half million points in total. The top selections were JSONB (13.7 percent), Security Simplification (11.1 percent), JCache (9.5 percent), Security Interceptors (8.2 percent), and MVC (model-view-controller; 7.8 percent).

In Parts 1 and 2 of the survey (PDF), developers identified how important they considered a wide range of potential Java EE 8 enhancements, mostly by responding to questions that could be answered with Yes, No, or Not Sure. The 13 enhancements featured in Part 3 of the survey were the ones that received 60 percent or more Yes responses in the first two parts.

Looking ahead to Java EE 8, **Linda DeMichiel**, Specification Lead for Java EE 7, says, "The next steps for us in this process are to evaluate this input, and balance it with feedback from our Java EE licensees, the members of our Expert Groups, customers, and our internal architects and Specification Leads. Invariably this will involve weighing competing opinions and balancing competing needs and objectives against the resources and time that we have to achieve them."

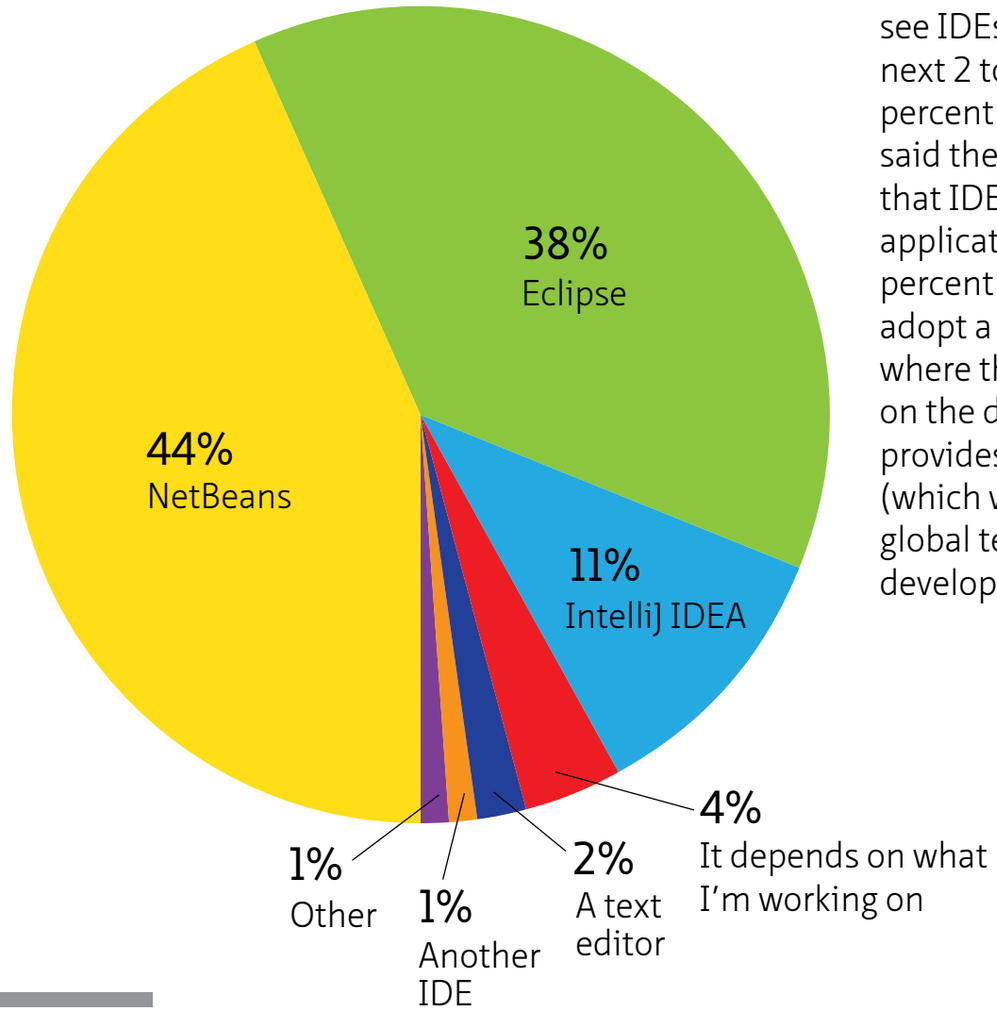
ART BY I-HUA CHEN

JAVA.NET POLL

Developers Prefer NetBeans, Eclipse, and IntelliJ IDEA

In a recent Java.net poll, 93 percent of developers indicated that they do most of their coding using NetBeans, Eclipse, or IntelliJ IDEA. The days of coding Java and Java Virtual Machine (JVM) languages using text editors are apparently behind us—with good reason! A group of 925 developers participated in the poll, which asked them to complete this sentence: "I do most of my coding using . . . ." Here are the results:

A separate poll asked, "How would you like to see IDEs evolve over the next 2 to 5 years?" 59 percent of developers said they would prefer that IDEs remain desktop applications, while 30 percent hoped IDEs would adopt a more hybrid form, where the IDE is available on the desktop but also provides a web interface (which would be useful for global teams or for when developers are traveling).



EVENTS

DEVOXX UK *JUNE 12–13*
LONDON, ENGLAND

Part of the Devoxx family, the conference is organized by the London Java Community. The organizers put developers front and center in the conference with their theme "Mind the Geek." The schedule is packed with presentations, in-depth hands-on labs, 15-minute lunchtime "quickies," and Birds-of-a-Feather (BOF) sessions. Developers will meet top British and international experts in a casual atmosphere.

PHOTOGRAPH BY BERIT WATKIN ON FLICKR

33rd DEGREE

JUNE 9–11
KRAKOW, POLAND
This conference, which is all about Java technology, features seven tracks over three days. Conference sessions, evening Birds-of-a-Feather (BOF) sessions, and three-hour-long workshops and labs led by world-class trainers are on the agenda.

QCon

JUNE 9–13
NEW YORK, NEW YORK
QCon is designed for technical team leads, architects, engineering directors, and project managers who influence innovation in their teams. QCon features a track focused on Java innovations, plus tracks on mobile, cloud, and much more. The first two days are tutorials, followed by a three-day conference.

JAVA EE SUMMIT 2014

JUNE 23–25
MUNICH, GERMANY
The Java EE Summit includes 14 intensive workshops about major Java EE topics led by top German-speaking Java EE experts. Developers and software architects will receive in-depth knowledge to plan and implement successful Java EE 6 and Java EE 7 projects.

FISL

JULY 3–6
PORTO ALEGRE, BRAZIL
The International Free Software Forum (FISL) event is jam-packed with lectures, workshops, demonstrations, and presentations.

ÜBERCONF EUROPE

JULY 7–10
BERLIN, GERMANY
ÜberConf, part of the No Fluff Just Stuff Software Symposium Series, explores the ever-evolving ecosystem of the Java platform. The conference offers 100 technically focused sessions including more than 25 hands-on workshops centered around architecture, cloud, security, enterprise Java, languages on the Java Virtual Machine (JVM), build/test, mobility, and agility.

JAVA FORUM 2014

JULY 17
STUTTGART, GERMANY
Java User Group Stuttgart (JUGS) organizes Java Forum, which is attended by more than 1,000 participants interested in Java and the Java environment. The forum includes lectures, presentations, workshops, demos, and Birds-of-a-Feather (BOF) sessions.

EVENTS

LONE STAR SOFTWARE SYMPOSIUM

JULY 18–20
AUSTIN, TEXAS
This conference focuses on the latest technologies and best practices emerging in the enterprise software development space. The Java sessions cover trends, best practices, and latest developments in Java application development. The conference features more than 55 sessions with leading industry experts, who share their practical and real-world experiences.

OSCON

JULY 20–24
PORTLAND, OREGON
Open source pioneers, experts, and innovators have gathered at OSCON over the past 16 years. With a dozen tracks and hundreds of sessions, the conference has practical tutorials, inspirational keynotes, and a wealth of information on open source languages and platforms.

THE DEVELOPER'S CONFERENCE (TDC)

JULY 30–AUGUST 3
SÃO PAULO, BRAZIL
One of Brazil's largest conferences for developers, IT professionals, and students, this event offers Java tracks over five days. Java-focused content includes introductory sessions in the Java University track, and advanced and intermediate sessions in the Java EE, mobile, and embedded tracks.

JAVA BOOKS

JAVA 8 POCKET GUIDE

By Robert Liguori and Patricia Liguori
O'Reilly, April 2014
Here's Java, stripped down to its bare essentials: possibly the only book on Java that you can actually fit in your pocket. If you've ever been writing code and gotten stuck because you can't remember how something works, the *Java 8 Pocket Guide* is an indispensable aid. This edition focuses on Java 8, including sections on lambdas and the Date and Time API. This book offers practical help for practicing developers.

JAVA PROGRAMMING INTERVIEWS EXPOSED

By Noel Markham
Wiley, February 2014
Java is virtually a requirement for businesses making use of IT in their daily operations. For Java programmers, this reality offers job security and a wealth of employment opportunities. But that perfect Java coding job won't be available if you can't ace the interview. If you are a Java programmer concerned about interviewing, *Java Programming Interviews Exposed* is a great resource to prepare you for your next opportunity. Author Noel Markham, an experienced Java developer and interviewer, has loaded his book with real-world interview examples.

MASTERING JAVAFX 8 CONTROLS

By Hendrik Ebbers
Oracle Press, July 2014
This Oracle Press guide shows how to create custom JavaFX 8 controls and master the development of rich clients and huge applications. It introduces JavaFX controls and the basic JavaFX APIs for handling them. It then reviews available controls and provides clear instructions on how to alter them as well as how to create new, custom controls specified to the user's needs. Developers new to JavaFX and those ready to start advanced work will benefit from this book.

JCP Executive Series

Java Tools for the Bottom Line

Individual Executive Committee member Werner Keil discusses the Java Money and Currency API, and his unique participation in the JCP. **BY STEVE MELOAN**

Werner Keil has worked for more than 25 years as a project manager, software architect, analyst, build manager, agile coach, and consultant for various Fortune Global 500 companies, including Oracle and IBM, across diverse industries, such as mobile/telco, Web 2.0, finance, insurance, travel, automotive, healthcare, media, and public services. Among his early clients was Sony, where he designed and implemented microformat-based tags for its online music portals.

Keil develops enterprise systems using Java and other technologies from Oracle, IBM, and Microsoft, and does web design and development using Adobe products, Ajax/JavaScript, and dynamic and functional languages.

Keil is a Committer at the Eclipse Foundation; a Babel

PHOTOGRAPHY BY TON HENDRIKS

Keil and Anatole Tresch, Specification Lead for JSR 354, talk with Stephen Chin for the NightHacking tour during the JavaLand conference in Brühl, Germany.

Language Champion; a UOMo Project Lead; and an active member of the Java Community Process (JCP), where he is a member of the Executive Committee. In addition, he is involved in many JSRs and open source projects, runs his own creative and talent agency—Creative Arts & Technologies—and writes song lyrics, novels, screenplays, technical books, and articles.

Java Magazine: Give us a quick overview of JSR 354, the Java Money and Currency API.

Keil: JSR 354 provides interfaces and classes designed to handle currencies and monetary data. There are four main areas of focus. The *Core* defines classes representing currency data and monetary amounts. The *Conversion module* provides APIs and service provider interfaces (SPIs) to convert from one currency to another, and it handles exchange rates between currencies. The *Format module* supports the parsing and formatting of currency data through a set of APIs and SPIs. And the *Extensions module* offers functionality such as regional support, validity services, and currency mapping.

The Money and Currency API targets all general application domains such as banking and finance, e-commerce, and mobile or embedded payments. It supports complex calculation rules, including regional algorithms and corresponding display precision. There is also logic for foreign exchange, complex usage scenarios, a plain-text representation of money, and printing and parsing to String.

We wanted to make the API very flexible with the ability to define custom currencies, either in a gaming environment or to accommodate the rise of virtual currencies.

Also, size was a key concern. This API needs to be applicable in the Java ME world, running on a money-changing box for instance, all the way up to a large server running Java EE. Scalability is hugely important. And I think we achieved that.

A FLEXIBLE API

We wanted to make the API very flexible with the ability to define custom currencies, either in a gaming environment or to accommodate the rise of virtual currencies.

Java Magazine: When processing money, security is paramount. Will you talk about that aspect?

Keil: Security is certainly a key issue. But the Java Money and Currency API, as such, is not responsible for this aspect. Monetary data must rely on other functionalities of the Java platform to provide a trusted environment. Java SE's security APIs handle cryptography, secure communication, authentication, access control, and public key infrastructure. Users and administra-

Keil and Oracle's Ed Burns look at Money and Currency API details during JavaLand.

tors are also provided a set of tools for securely managing applications. I was involved in JSR 321, the Trusted Computing API for Java, which was designed to access Trusted Computing functionality from Java applications. Members of the Executive Committee have to be cognizant of usage scenarios to ensure that financial transactions cannot be intercepted or duplicated. These are crucial issues in the security landscape.

Java Magazine: Are lambda expressions supported by the Money and Currency API?

Keil: The Money and Currency API is not directly tied to the APIs of Java 8. Many enterprise customers will use the JDK that is included with their applications server, and for a while that will be Java SE 7. So we have to be backward compatible. But we use many of the Java 8 functional interfaces in a pattern and signature that will be compatible. Lambda expressions built into the API, however, will probably not be feasible until at least Java SE 9.

Java Magazine: Tell us about your involvement with the JSR 354 process. And what is the current status of the API?

Keil: I was influential in the genesis of this JSR. It first came up during a panel discussion at QCon in London seven years ago. There was general agreement that a JSR targeted at money and currency would be very useful. And we talked about some of the logistics regarding how it would mesh with future Java releases.

Five years ago, when I was already a member of the Executive Committee, we had a face-to-face meeting in Princeton, where I did a presentation on units of measurement in support of JSR 275. While that effort was stopped at the JCP phase, parts of it inspired subsequent work on JSR 354.

Credit Suisse joined the Executive Committee soon after that, and understood the value of a currency and money JSR. Anatole Tresch, from Credit Suisse, stepped in as Specification Lead and has done a fantastic job. We expect Public Review 2 in the second quarter of this year. If that passes, and the JSR is viewed as feature-complete, a final

REAL-WORLD API

This API needs to be applicable in the Java ME world, running on a money-changing box for instance, all the way up to a large server running Java EE.

Tresch and Keil take a break between sessions at JavaLand, where they were both speakers.

JSR 354 TIMELINE

We expect Public Review 2 in the second quarter of this year. If that passes, and JSR 354 is viewed as feature-complete, a final version might be ready around the JavaOne time frame.

version might be ready around the JavaOne time frame.

Java Magazine: How does your involvement with the Eclipse Foundation influence your work within the JCP?

Keil: JSR 275 was a general-purpose measurement effort. But at the time, back in 2010, it was viewed as a too-small niche for the Java platform. So as Specification Lead, I pursued the concepts in an open source project called unitsofmeasurement.org, which is a library that provides Java interfaces for handling units and quantities. The project design benefited from comments from Executive Committee members—some of whom voted against JSR 275. So the new work is improved and is different from the old JSR. The revitalized project was implemented by an open source project called UOMo under the Eclipse umbrella, and is led by me. It allows developers to convert between various systems of measurement, and to format or parse string representations of units.

A couple of months ago, I discussed reviving a JSR to ensure that the work we've done with unitsofmeasurement.org and UOMo can be used in a new JSR that will add value to Java, for example, making it comparable to languages such as C/C++ in this regard. This was proposed to the JCP Program Management Office, and has now been filed as JSR 363. I think this JSR will be particularly useful for embedded Java applications supporting the evolving Internet of Things space, which will spawn unbelievable volumes of data that need interoperability.

So, this is an excellent example of how the JCP and open source efforts, such as those of Eclipse, can interact and add value for all the parties involved. It's a vibrant ecosystem that produces innovation.

Java Magazine: You are the only independent member of the JCP Executive Committee. How is your involvement unique? And what motivated you to join?

Keil: In 2008, a few months after I left BEA as an independent contractor, I became a candidate for the Executive Committee. I got the most votes of all the individual candidates on the Java SE side, thanks to my broad experience and my history of participation in JCP activities. I had been a member of a couple of Expert Groups and was asked to help JSR 275 as one of the Specification Leads.

My varied experience as an independent contractor gave me a unique view of the Java landscape from many different angles, ranging from Java EE

and large portals to embedded Java on small devices. I try to bring these experiences into my Executive Committee participation.

I originally joined the JCP because I wanted to contribute toward making the Java platform as robust and competitive as possible. We all depend on these tools to bring our ideas to fruition, so we need to be sure that the best solutions become part of the technology.

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

Keil: JSR 358, with Oracle's Patrick Curran as Specification Lead, makes a number of important revisions and adjustments to the JCP. Revising the Java Specification Participation Agreement [JSPA] is a major focus for this JSR, which should have a positive impact on JCP participation. Sometimes employers are reluctant to participate, because the JSPA has intellectual property and patent implications that can be problematic for some companies.

One possibility is to follow the lead of the Eclipse Foundation and allow developers to get involved as individuals, apart from their employer, with a designation of affiliate or associate. This would involve a separate agreement that typically allows a developer to contribute ideas without interacting directly with code. This type of membership should have a positive impact by simplifying the legal agreement and the role. The goal is to avoid any patent or intellectual property disputes that might arise from work done for the JCP.

JCP.next is a group of JSRs (including JSR 358) dedicated to improving JCP processes. We hope to increase participation by individuals who are self-employed, or are members of a Java user group [JUG], or work at a university. And we also plan to improve participation through the use of open source development processes and public mailing lists.

Java Magazine: Any closing remarks?

Keil: I hope the Executive Committee will always have some independent members, individuals, and/or JUGs. More diversity of background and experience increases innovation and problem resolution. And that contributes to the overall quality and viability of the Java platform. `</article>`

Keil talks with Terrence Barr, senior technologist and product manager at Oracle.

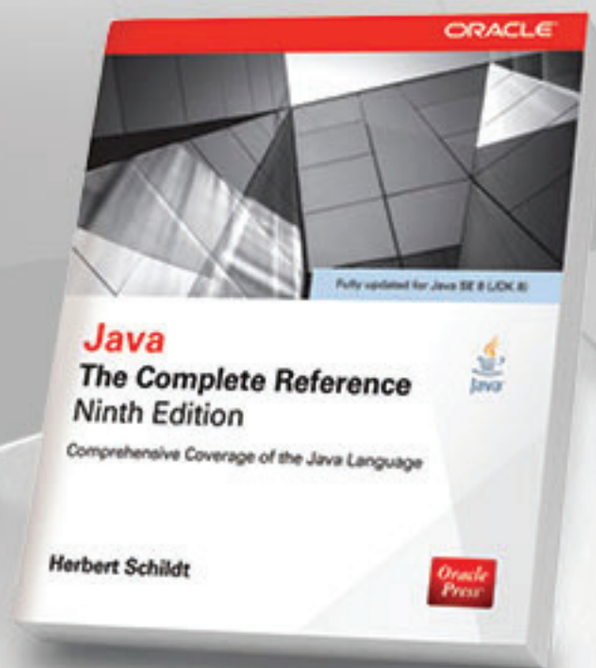
Steve Meloan is a former C/UNIX software developer who has covered the web and the internet for such publications as *Wired*, *Rolling Stone*, *Playboy*, *SF Weekly*, and the *San Francisco Examiner*. He recently published a science-adventure novel, *The Shroud*, and regularly contributes to *The Huffington Post*.

LEARN MORE

- JSR 354: Money and Currency API
- JSR 354 project page on Java.net
- NightHacking interview

Oracle Press™ Your Destination for Java Expertise

Just in time for the Java SE 8 launch, these new Oracle Press books offer the most definitive, complete, and up-to-date coverage of Java available.



Java: The Complete Reference, Ninth Edition
Herbert Schildt

Fully updated for Java SE 8, this definitive guide explains how to develop, compile, debug, and run Java programs.

Java: A Beginner's Guide, Sixth Edition
Herbert Schildt

This fast-paced Java programming tutorial is revised to cover Java SE 8.

Mastering JavaFX 8 Controls
Hendrik Ebbers

Get a thorough introduction to JavaFX controls and the basic JavaFX APIs to handle them.

Quick Start Guide to JavaFX
J.F. DiMarzio

Create and deploy dynamic enterprise client applications in no time.

Oracle Press™

Available in print and as eBooks

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

Pinkmatter Cofounder and Chief Software Architect Chris Böhme at the South African National Space Agency satellite receiving station

TRANSFORMING DATA INTO INFORMATION

A portfolio of Java applications from South African software house Pinkmatter Solutions helps turn seemingly unrelated data into useful information. **BY PHILIP J. GILL**

PHOTOGRAPHY BY JAMES OATWAY/GETTY IMAGES

Pretoria, South Africa–based software house Pinkmatter Solutions boasts a diverse product portfolio: there's Maltego, an open source intelligence and data gathering suite; FarEarth, a satellite image processing, enhancement, and analysis tool; and Phoenix, a text mining and document linkage engine. At first glance, these solutions seem to have nothing in common, except that they are all written in Java.

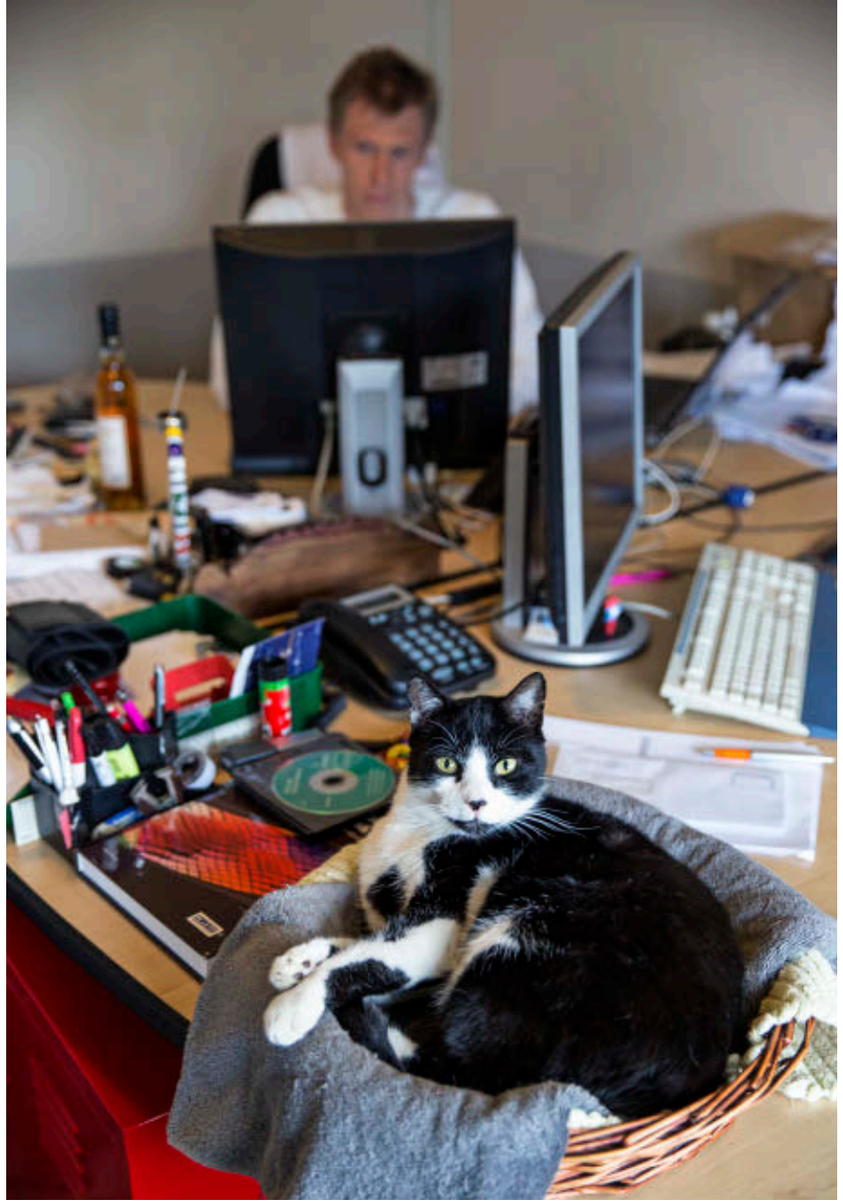
PINKMATTER SOLUTIONS
pinkmatter.com

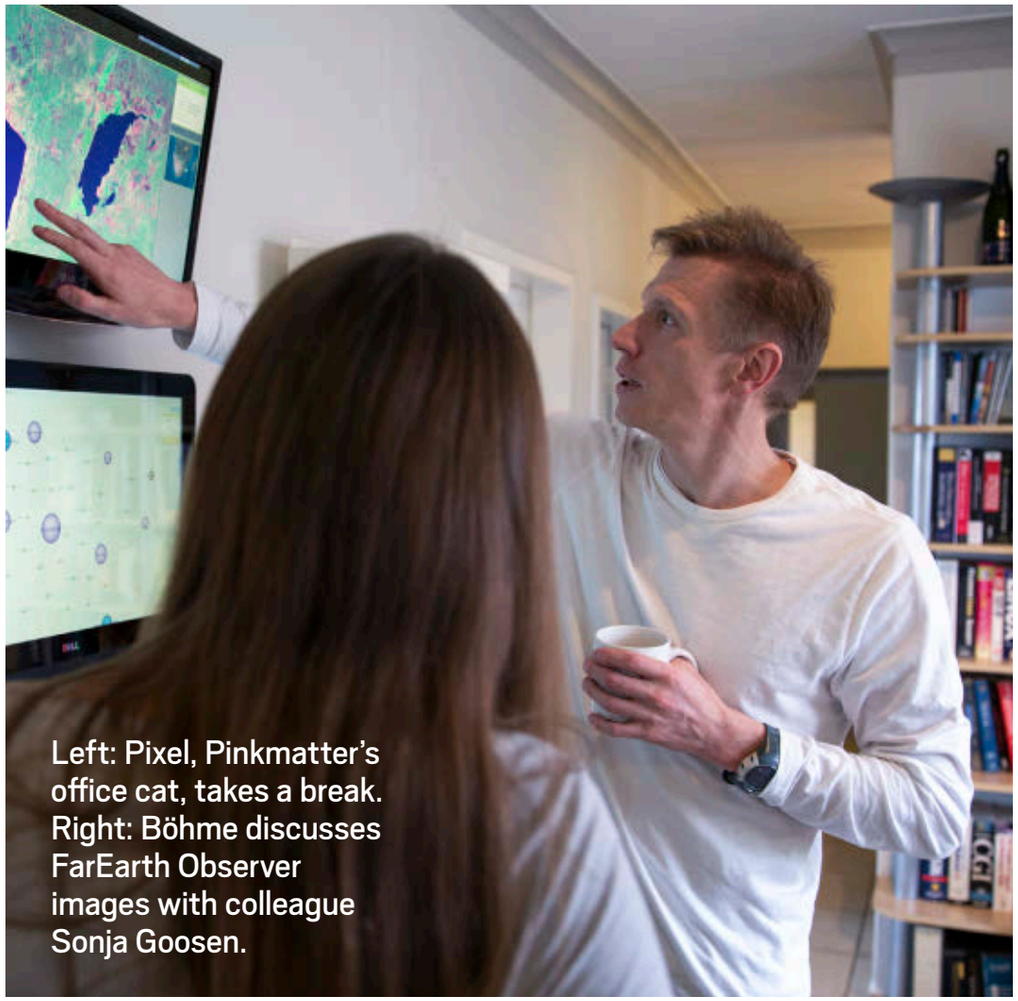
Location:
Pretoria, South Africa

Industry:
Software

Employees:
8

Java technologies used:
JDK, the NetBeans Platform, NetBeans IDE





Left: Pixel, Pinkmatter's office cat, takes a break. Right: Böhme discusses FarEarth Observer images with colleague Sonja Goosen.

On closer inspection, however, it becomes clear that for all their differences, these applications have a common purpose—helping users make connections between seemingly random pieces of data, linking them by group, type, or other characteristics, and then presenting them graphically in ways that can reveal potentially valuable information.

"On the internet, there's lots of information—names, IP addresses, e-mail addresses, and so on—that in isolation don't tell you very much," explains Chris Böhme, Pinkmatter's cofounder and chief software architect. "But when you discover links between them and present them in a graphical way, it can get very interesting." Böhme notes that Maltego does just that—no illegal snooping or invasion of privacy needed.

"The information is out there in the open," Böhme continues. "Customers can of course add their own information, but when we call Maltego an 'open source intelligence' tool, we mean the information that it uses is openly available in public sources on the internet, not that it's licensed as open source software."

As an example, Böhme mentions a somewhat notorious celebrity socialite who frequently posts images on Twitter. "It's pretty easy to find her home address in just a few mouse clicks based on the pictures she tweets, because there are GPS coordinates inside those pictures" he says. "In five mouse clicks, you can have her home address."

A Maltego screenshot shows tweets and followers of Paterva.

While that may be fun for fans and the paparazzi, law enforcement and businesses around the world also use Maltego for far more serious matters. In the banking industry, for example, Maltego is used as a computer forensics tool to help ferret out fraud. "Maltego can link internet IP addresses to bank accounts, so that in the case of suspected bank fraud or money laundering, it can graphically depict where money was deposited and what account received money from that account, and follow the trail," says Böhme.

Maltego, written in Java on the NetBeans Platform, uses what the company calls "transforms" to link one type of information item—names, companies, e-mail or IP addresses, websites, and so on—to another and to present those links graphically in diagrams that resemble computer network schemas. "Maltego uses a form of type matching and merging of entities," says Böhme. "There's no AI or learning algorithm in that sense."

Maltego is developed for another South African company, Paterva, which markets, supports, and has managed to establish the product worldwide. Maltego licenses are currently in use in a variety of industries for intelligence and security applications, and a free community edition with limited functionality currently has 180,000 registered users.

PICTURING CHANGES

After successfully building Maltego, the Pinkmatter team looked around for another opportunity and found one in satellite image processing and enhancement. Their second product, called FarEarth, is a suite of Java applications that is also built on the NetBeans Platform and is aimed at space agencies.

The first component in the FarEarth suite was originally created for the South African National Space Agency, and

NetBeans IDE 8 Arrives

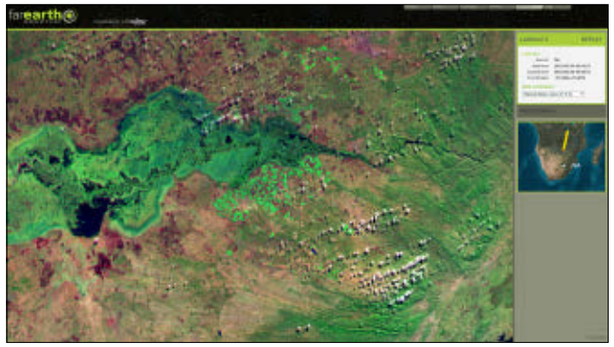
Oracle has released NetBeans IDE 8, the newest version of the popular Java open source integrated development environment (IDE). NetBeans 8 features improved support for Maven, Java EE with PrimeFaces, PHP, and C/C++, and new tools for HTML5, including AngularJS.

"NetBeans 8 has a number of nice updates to its code editing features," says Chris Böhme, cofounder and chief software architect at Pinkmatter Solutions in Pretoria, South Africa. Pinkmatter's Maltego solution, an open source intelligence and security matrix suite, makes use of the new NetBeans functionality—which includes new out-of-the-box code analyzers and editors. "We use the code editor inside Maltego so that users can author macros called Maltego Machines that can automate many tasks users would otherwise do by hand," says Böhme, offering automatic monitoring and footprinting of an organization's infrastructure or monitoring activity on Twitter as examples.

"NetBeans 8 is more than an IDE; it is also an application framework known as the NetBeans Platform," adds Geertjan Wielenga, principal product manager for NetBeans at Oracle. "The NetBeans Platform is a generic application framework for Java desktop applications that provides the infrastructure plumbing that every developer needs out of the box, so that they don't need to manually code these or other basic features themselves."

NetBeans 8 was released simultaneously with JDK 8 and works seamlessly with Java SE 8, Oracle Java SE Embedded 8, and Oracle Java ME Embedded 8.

Below, from left: Pinkmatter's Goosen, Böhme, Anton van Aswegen, and Philip Bouwer walk at the satellite receiving station.



A FarEarth Observer screenshot shows swamps near Lake Manyeke in Southern Zambia as seen by the Landsat 8 satellite.

today four other countries—Nigeria, Indonesia, Germany, and Australia—use FarEarth components for applications such as environmental monitoring. "Images produced by the FarEarth system are used in forest monitoring, agriculture, and water management," says Böhme. "In Indonesia, for instance, there's a big push to get as many images as they can to see how their rain forests degrade and recover over time."

Böhme mentions in particular a United Nations–sponsored project in developing countries called REDD, which is short for Reducing Emissions from Deforestation and Forest Degradation. REDD gives money to countries to look after their rain forests, "to help preserve and restore them," he explains. "Some of our customers use FarEarth images to apply for funds under the REDD program."

The FarEarth system receives raw satellite image data in real time and renders it one line at a time. "The color bands are not aligned when you get them, so what you see is smudges," says Böhme. "Through many, many steps, we create a clear image by applying geographic and geometric corrections and aligning the eight different spectral bands—so that all the colors look normal."

The next step in the process is to take raw Landsat images of the same land areas over time and then compare and analyze those images against each other for changes on the ground, identifying potential threats to greenbelts and forests.

Once processed and enhanced through FarEarth, the images are stored and cataloged for later retrieval and analysis by the public, scientists, and government departments and agencies.

"You can look at each pixel over the years and use that as a scientific measurement," says Böhme. "From these images you can see actual changes; in the winter the trees in a forest are brown and dark, and in the summer they're light green, for example. You can also look at images of the same land area from year to year and see what changes have

MAGICAL NAMESAKE

The precursor to Maltego was named "Palantir," after a magical object in the film *The Lord of the Rings*.

Böhme at his office in Pretoria, South Africa

occurred on the ground by analyzing the differences in the images. To have all this done fully automated is what we are working towards."

TEXT AND BIG DATA

A third Pinkmatter product, called Phoenix, is currently in development, with a beta version expected later this year. Written in Java, Phoenix employs a full suite of Apache open source technology for big data applications, including Hadoop, HBase, and Elasticsearch.

CLIMATE DEFENDER

The United Nation's REDD program has spent more than US$170 million fighting climate change in developing nations.

"Phoenix does big data mining on large volumes of documents and then visualizes how the information in those documents fits together," says Böhme. "It uses a functionality we call 'disambiguation.' If you have one document that mentions John Smith and another mentions John Smith, how do you know if it's the same John Smith or not?" With Phoenix, Böhme explains, "We scan the documents and build a database of the information. We have learning models that actually decide whether this John Smith is the same or different."

A typical application of Phoenix might take place in a law firm or the legal department of a large corporation. "From a bunch of legal documents, we extract information and create a Maltego-like graph that would show who owns what, for example," Böhme says. "All that information is automatically learned and extracted from the documents without human beings having to read thousands of pages."

For all these applications, Böhme says, Java was the only choice. "I had programmed in C#, Java, C++, and other languages back in 'the good old days' when I was doing cryptography," Böhme recalls of the years before he started Pinkmatter with cofounder Sonja Goosen. "But Java was always my favorite language, and I fell in love with it from the start."

In addition to being his favorite, Java had two technological advantages—and still does—over other languages such as C#, says Böhme. The first is platform independence.

"At the time we started Pinkmatter, people talked about C# that way, but that was really a pipe dream," he says. "The only strongly typed language that gave us real platform independence on the Mac, BSD, Linux, and Windows was Java, and it still is."

The second advantage is the NetBeans Platform. "Having an application framework that gives you auto-updates, runtime composition, loadable modules, a wonderful window management framework, and other important features right out of the box wasn't matched by Microsoft and C# then and still isn't," notes Böhme. `</article>`

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

MICHAEL KÖLLING

BIO

Part 1

Interactive Objects with BlueJ

Interactivity and visualization help beginners learn and form mental models.

When you talk with people about learning to program, and especially learning to program in Java, the BlueJ environment is often mentioned as a good introductory environment to get started. And with good reason: BlueJ is an excellent environment in which to gain a good understanding of fundamental principles of object-oriented programming. When asked why beginners should not just start by using Eclipse or NetBeans—environments known for their excellent toolset and great functionality—the answer typically points to the great value of BlueJ's simplicity and interactivity for gaining a thorough understanding of programming principles.

But what does this really mean?

When talking about BlueJ, it is not immediately clear what we mean by *interactivity*, nor is it clear why that matters. In this two-part series of articles, we will explore and discuss BlueJ's interactive features in detail to see what they can do for you and why you should care.

BEYOND SYNTAX

Learning to program is not about syntax. Well, at least not mainly about syntax. All of us who have been around programming for a while know that syntax is neither the interesting nor the hard part.

It's Not About Syntax

Learning to program is not about syntax. Well, at least not mainly about syntax. Learners new to programming often struggle to get their semicolons in the right places, match their curly brackets, or remember the syntax of a for loop. They often think that this is what programming is about: learning to get the syntax right. But of course it isn't.

All of us who have been around programming for a while know that syntax is neither the interesting nor the hard part. In fact, every student at the university where I teach will eventually learn the syntax of their first programming language. It seems hard to some of them at first, but it's actually the easy part.

No; programming—in this case, object-oriented programming—is about learning how to translate your problem into a program structure, how to design your object interactions, and how to structure your classes. It is about designing algorithms and interacting object structures. And this is hard.

To do this effectively, it is crucial to have a well-developed mental model of object interactions—both how they work in general and how they are arranged in the particular program you are working on.

And this is the crucial difference between beginners and experts: experts have a well-developed mental model of the fundamental principles of object orientation, while beginners do not. Beginners need to *develop* this mental model, and an educational programming environment must help learners do so.

Environments for Experts, Environments for Learners

Environments designed for professionals—NetBeans, Eclipse, and the like—work very well for experts. They provide a great toolset for programmers who know what they are doing, but they do little to help learners develop the mental models they need

Development Tools and Techniques

PHOTOGRAPH BY JOHN BLYTHE

to become good programmers.

The main focus in these environments is on source code. Beginners using these systems stare at lines of code, so it is not surprising that what they think about is lines of code. Yet, what is really much harder, and much more important, to learn are the fundamental abstractions: classes, objects, and interactions. These are not well represented in professional environments. Objects, for example, hardly exist as a tangible abstraction in a typical interface. Students look at lines of Java code and struggle to understand the difference between classes and objects.

This is not a problem for practiced programmers, because they have a working model in their minds. However, for beginners this is a fundamental problem—a much harder and more fundamental one than the question of where the semicolons go—and they get no help from the environment.

An educational development environment must help learners develop the mental models needed, and it must put the important abstractions—classes, objects, and method calls—in front of the students' eyes and let them think and talk about them.

And this is exactly what BlueJ does. That is why starting with BlueJ will give beginners a stronger foundation in programming principles. Once they have developed a good working model of object-oriented program execution, they are ready to leave BlueJ behind and switch to a professional environment. But a programmer without a strong foundation in the fundamental principles is a dangerous thing.

STARTER KIT

BlueJ is designed to offer just the right toolset for beginning programmers. It appears simple, **yet does sophisticated things** to help them work and learn.

A Few Words About Simplicity

At the beginning, we mentioned simplicity and interactivity as the two important points. Simplicity is the easier one to discuss; it is easy to see, and most people get the point quickly.

Professional environments offer many tools that are helpful, and often essential, for experts, but these can easily become a hindrance, rather than help, for learners. Beginners use only a small fraction of these tools, but they initially don't know *which* fraction they need to use. Finding the right workflow is made harder by the presence of unwanted options, and orienting yourself in the environment is distinctly difficult on your first encounter.

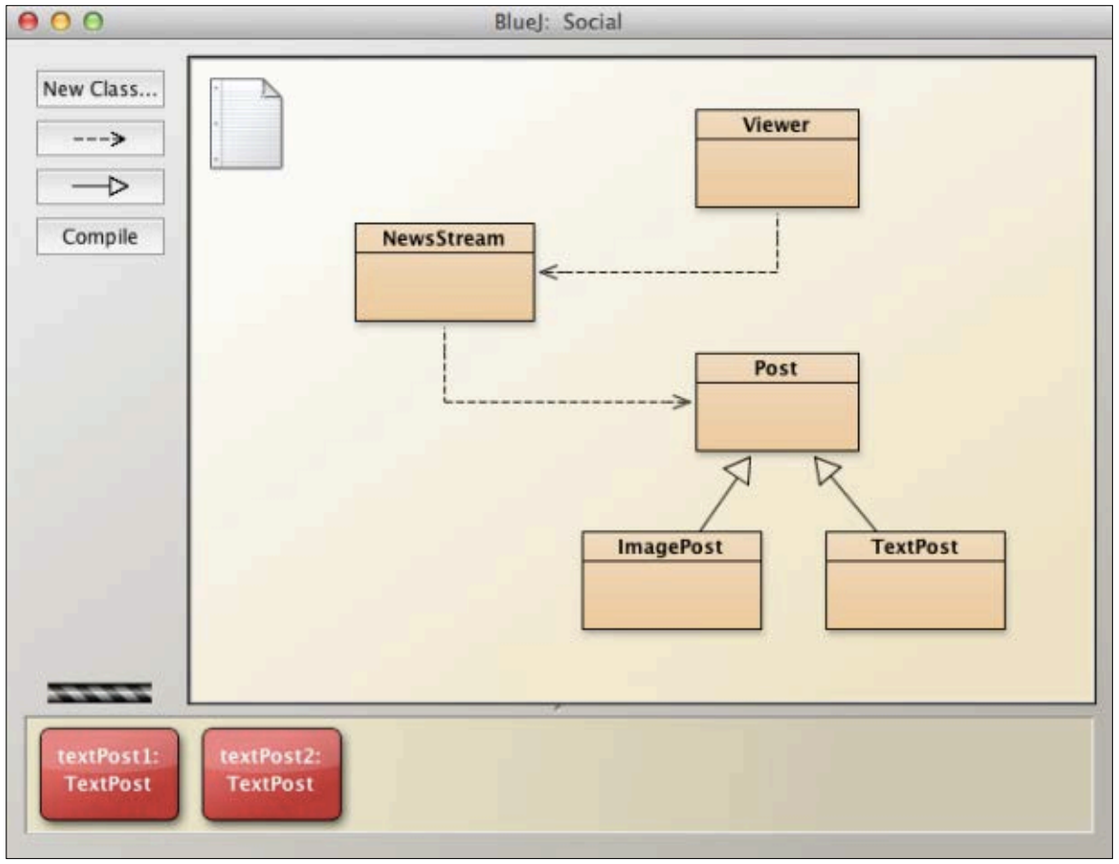


Figure 1

Choice, for people who don't need it, is a bane, not a favor.

BlueJ is designed to offer just the right toolset for beginning programmers. It appears simple, yet does sophisticated things to help them work and learn. After only a short time, users feel comfortable in the environment and feel they know it well, and BlueJ blends into the background. Learners don't need to think about the environment; they think about programming principles. The development environment becomes a silent helper, not a problem to be mastered in itself.

BlueJ's interface (see **Figure 1**) appears very simple and unthreatening. This should not be mistaken as simplicity of functionality: BlueJ offers some sophisticated tools that help learners along the way, and we will discuss some of them shortly. In fact, one of the greatest achievements of BlueJ is to present a complex problem domain—programming—in a simple fashion. One of the leading design

goals of BlueJ is well expressed in a famous quote by Antoine de Saint Exupéry:

"Perfection is achieved, not when there is nothing more to add, but when there is nothing left to take away."

But it is also important to note that this does not mean that an educational environment is simply a cut-down version of a professional one. BlueJ's toolset is not a subset of one of the bigger systems. It is not merely about offering *fewer* tools, but about offering *different* ones.

BlueJ has various bits of functionality not found in many other environments, and most important among them is a greater level of interactivity and visualization. These are the important tools for learning to program and forming your mental models, and, therefore, they will be the focus of the rest of this article.

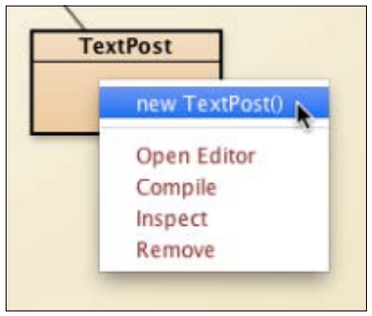


Figure 2

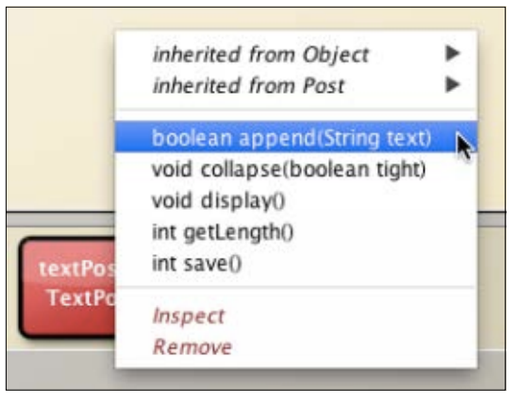


Figure 3

The Killer Features: Interaction and Visualization

A previous issue of *Java Magazine* introduced BlueJ's features by presenting a small programming example, and another issue discussed testing in BlueJ in more detail.

Both of these articles touched on BlueJ's interactive features and made use of some of them. However, they presented only a partial insight into BlueJ's toolset. We will now discuss BlueJ's interaction features in more depth—the first few here, and the rest in Part 2.

Classes and Objects

The first, most obvious—and most fundamental—aspect of BlueJ's functionality that is relevant here is its presentation of classes and objects.

When opening BlueJ, the first thing shown to a user is a diagram of the classes in the project and their relationships. The diagram's arrows display inheritance and *uses relationships*—which is when a class references another.

A user can then interactively create objects by right-clicking any of the classes and invoking a constructor from the class' menu (see **Figure 2**). Every constructor of the class is shown in this menu and can be interactively invoked. Objects, once created, are shown in red on the object bench along the bottom of the main window (see **Figure 1**).

Once an object has been created, a menu offers all the object's public methods, and any of them can be interactively invoked by selecting it from this menu (see **Figure 3**). If the method has parameters, they can be entered; otherwise, the method executes immediately.

GET VISUAL

BlueJ has various bits of functionality not found in many other environments; **most important among them is a greater level of interactivity and visualization.** These are the important tools for learning to program and forming your mental models.

Why Is This Important?

Even the first few simple visualizations and interactions shown so far have a fundamental effect on the understanding of the principles of object orientation.

First, the division of a program into classes is illustrated. The diagram conveys the message that an object-oriented program is represented by a set of cooperating classes, and that these parts of a program can be examined and understood separately. Presenting the classes in a diagram illustrates their relationship better than the simple list used in many other environments.

Second, the difference between classes and objects is implicitly apparent. Understanding this difference is one of the great problems for beginning students. Most teachers have struggled with this. In traditional environments, students start by looking at source code. What do they see—a class or an object? Well, the answer is neither or both. Which one it is depends on context: whether the student thinks about the static or dynamic properties of their program. Understanding the difference is difficult.

In BlueJ, the distinction is immediately clear: from a class, we can create objects. In fact, we can create as many objects as we like. In BlueJ, the question never comes up. Students get a feeling for classes

and objects almost automatically, with little effort. Both the visualization and the interaction (the act of creating the object) support this understanding.

Third, it becomes immediately clear that we can communicate with objects by invoking their methods. Students can experiment with objects, try out what they do, gain experience with parameters and return values, and investigate a project or parts of a project. The understanding developed through these activities will be very useful once students start writing code.

Understanding object orientation is about understanding three big concepts that define objects: state, behavior, and identity. Even these first simple interactions already illustrate two of them: behavior and identity. (We will soon see how state is visualized.)

No Test Drivers

One of the important aspects to emphasize about BlueJ is ease of interaction: methods can be executed individually without the need to write any test drivers. No public static void main is necessary. This means that classes and objects (and also students) can become active before writing code. Easier and quicker interaction means more interaction. More interaction means better understanding.

The possibility of interaction fundamentally changes how a learner can approach gaining an understanding of programming principles.

THE BASICS

BlueJ is an excellent environment in which to gain a good understanding of fundamental principles of object-oriented programming.

Conclusion

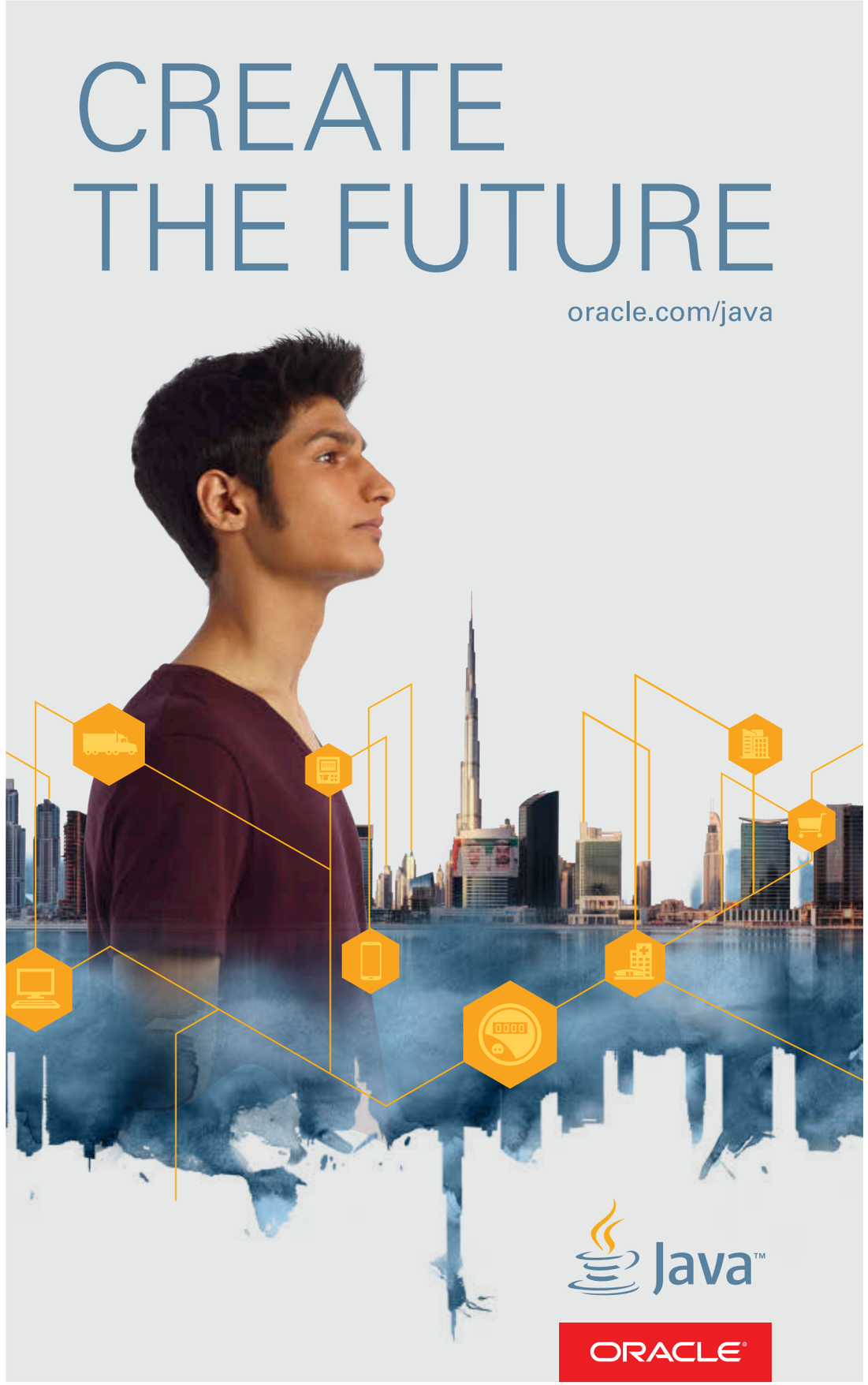
The use of visualization and interaction leads to the formation of better and stronger mental models that then form the basis for reasoning about programs and programming. So far, we have seen only a small example of BlueJ's visualization and interactive features. In the next article, we will discuss many more tools that support this process. `</article>`

MORE ON TOPIC:

Development Tools and Techniques

LEARN MORE

- Download BlueJ for free
- Example projects at the BlueJ book website



DEVELOPER TOOLS AND TRENDS

Oracle's **Chris Tonas** talks about Java tools and development frameworks.

BY TORI WIELDT

PHOTOGRAPHY BY BOB ADLER/GETTY IMAGES

Development Tools and Techniques

Nothing starts a flame-fest like a discussion of IDEs. Developers have religious-like fervor about the best IDE to use—or whether to use an IDE ("Emacs is all I need!"). It's obvious that one size doesn't fit all, but what tools can help most Java developers? What is Oracle's role in developer tools as the steward of Java and

Chris Tonas (left) and Sundar Sarangan, member of the technical staff at Oracle, talk about development for small devices and tablets.

NetBeans IDE and as the creator of Oracle JDeveloper? What are the significant trends that are shaping software development and tools?

Java Magazine *sat down with Oracle's vice president of mobility and application development tools, Chris Tonas, to hear his take on the latest in the world of Java tooling and development frameworks.*

Java Magazine: Can you tell us a little bit about your role at Oracle as it relates to development tools?

Tonas: I lead the organization that is working on Oracle's software development tools and frameworks, specifically, the teams that build our offerings for Java developers—whether in NetBeans, Eclipse, or Oracle JDeveloper. Our team also builds the tools and frameworks that are used by developers working with Oracle's cloud and mobile platforms.

Java Magazine: March 2014 saw the release of JDK 8 and with it, NetBeans IDE 8. How do you view this release?

Tonas: The release of JDK 8 and NetBeans IDE 8 represents a big step forward for both Oracle and the Java community. A lot of hard work and collaboration went into this milestone, and I'd like to take a moment to thank everyone who contributed to this achievement.

Java Magazine: With the new NetBeans IDE 8.0 out, what are the plans for NetBeans going forward?

Tonas: In the short term, an update release of NetBeans IDE 8 is under way to align with Java ME 8. Additional NetBeans 8 releases that target specific bugs are anticipated to be released after that. Longer term, Oracle is committed to the continued success of both Java and NetBeans. Work on JDK 9 is now under way and we're planning a NetBeans 9 release to go along with that, as usual.

Java Magazine: As you mentioned, Oracle supports more than just NetBeans IDE. What's the thinking behind that?

Tonas: Oracle recognizes that developer tools aren't a one-size-fits-all proposition. Oracle is a significant contributor to the Eclipse project, and we are continuing to extend the capabilities of our Eclipse-based solutions as well. We offer Oracle JDeveloper for those who want the tightest alignment with the Oracle Fusion Middleware stack. In addition, we recognize that many JavaScript developers want to use lightweight tools, and we are planning to address those needs as well.

Java Magazine: What are some of the key trends you see in the software development space right now?

Tonas: It's clear that several significant trends are shaping software development and tools. Oracle is at the forefront of these changes and a leader in almost every aspect. We see three main changes happening right now.

First, Java remains the industry standard for server-side development, but we see growing demand to support developers using the combination of JavaScript and HTML5 for the presentation layer. We see JavaScript starting to gain ground for some server-side use cases as well.

Second, the shift to cloud-based deployment is now mainstream. Development for the cloud presents a new set of challenges and demands a fresh approach.

The third shift is the move to mobile. Mobile development must be integrated across the enterprise

IDEs at a Glance

Eclipse
From: Eclipse Foundation
Price: Free
Latest release: Eclipse Kepler (4.3.2)
License: Eclipse Public License

The Eclipse ecosystem is rich with projects and plugins. Java developers can choose from several packages, including Eclipse Standard, Eclipse IDE for Java Developers, Eclipse IDE for Java EE Developers, and more. These packages include tools for Java developers creating Java EE and web applications, including a Java IDE, tools for Java EE, Java Persistence API (JPA), JavaServer Faces (JSF), Mylyn, EGit, and others.

IntelliJ
From: JetBrains
Price: Community Edition—free; Ultimate Edition—US$500 (application server and frameworks support, database support, additional languages)
Latest release: IntelliJ IDEA 13
License: Commercial

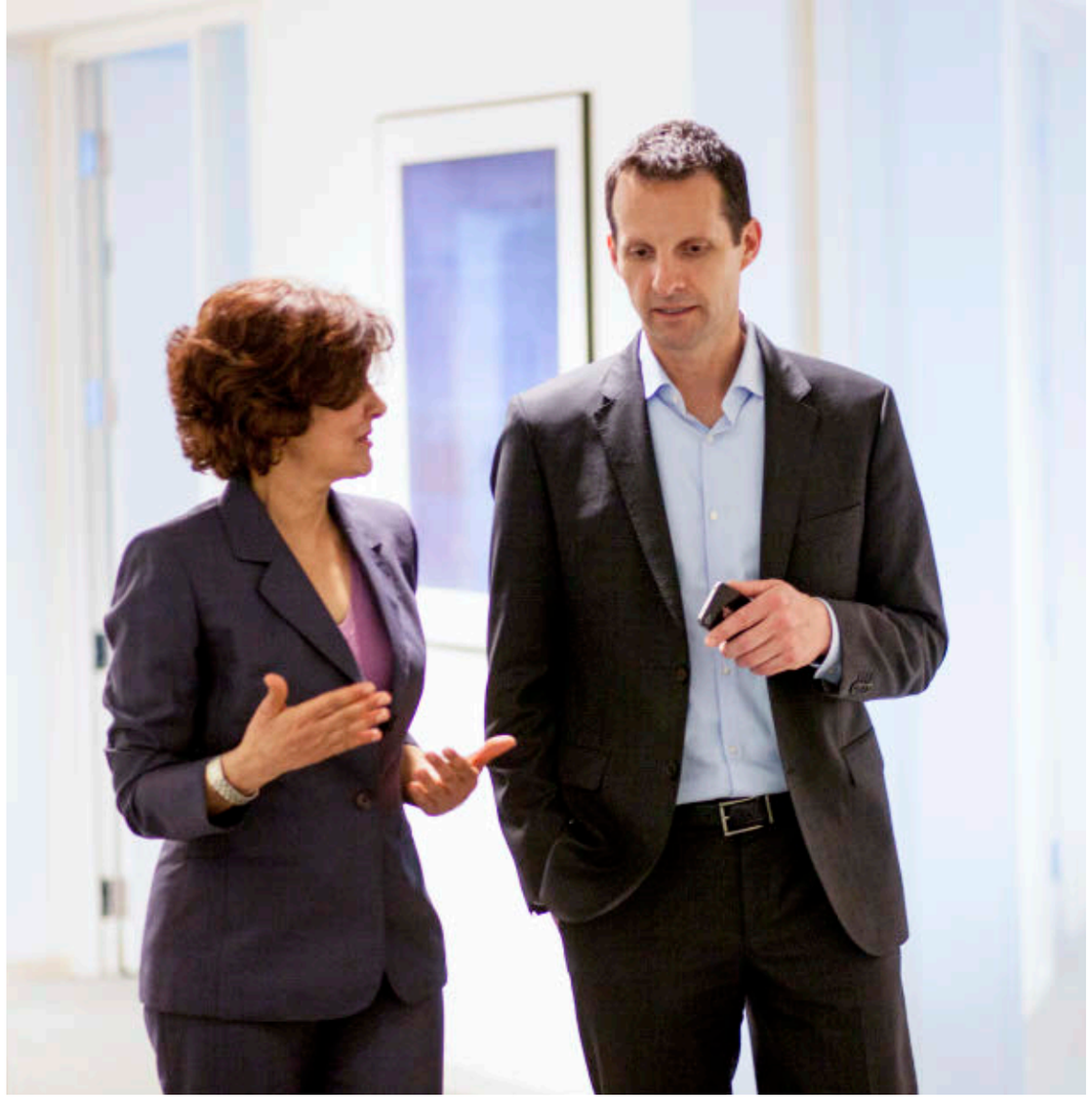
IntelliJ IDEA provides a comprehensive feature set including tools and integrations with the most-important modern technologies and frameworks for enterprise and web development with Java, Scala, Groovy, and other languages.

NetBeans
From: Oracle
Price: Free
Latest release: NetBeans IDE 8.0
License: A dual license consisting of the Common Development and Distribution License (CDDL) v1.0 and the GNU General Public License (GPL) v2.

Ready to go out of the box, NetBeans IDE provides code analyzers and editors for working with the latest Java technologies. NetBeans IDE supports many languages, including PHP, JavaScript, HTML5, Groovy, and C++.

Oracle JDeveloper
From: Oracle
Price: Free
Latest release: Oracle JDeveloper 12*c*
License: OTN JDeveloper License Agreement

Oracle JDeveloper is the development environment for Oracle Fusion Middleware with specific extensions that cover Oracle SOA Suite, Oracle WebCenter Portal, and Oracle Business Intelligence 11*g*. It integrates a full database development environment for both Oracle and non-Oracle databases and offers a complete solution for developers looking to increase their productivity building SOA-enabled enterprise Java applications.



Pari Tajiki, director of development at Oracle, catches up with Tonas about product release schedules.

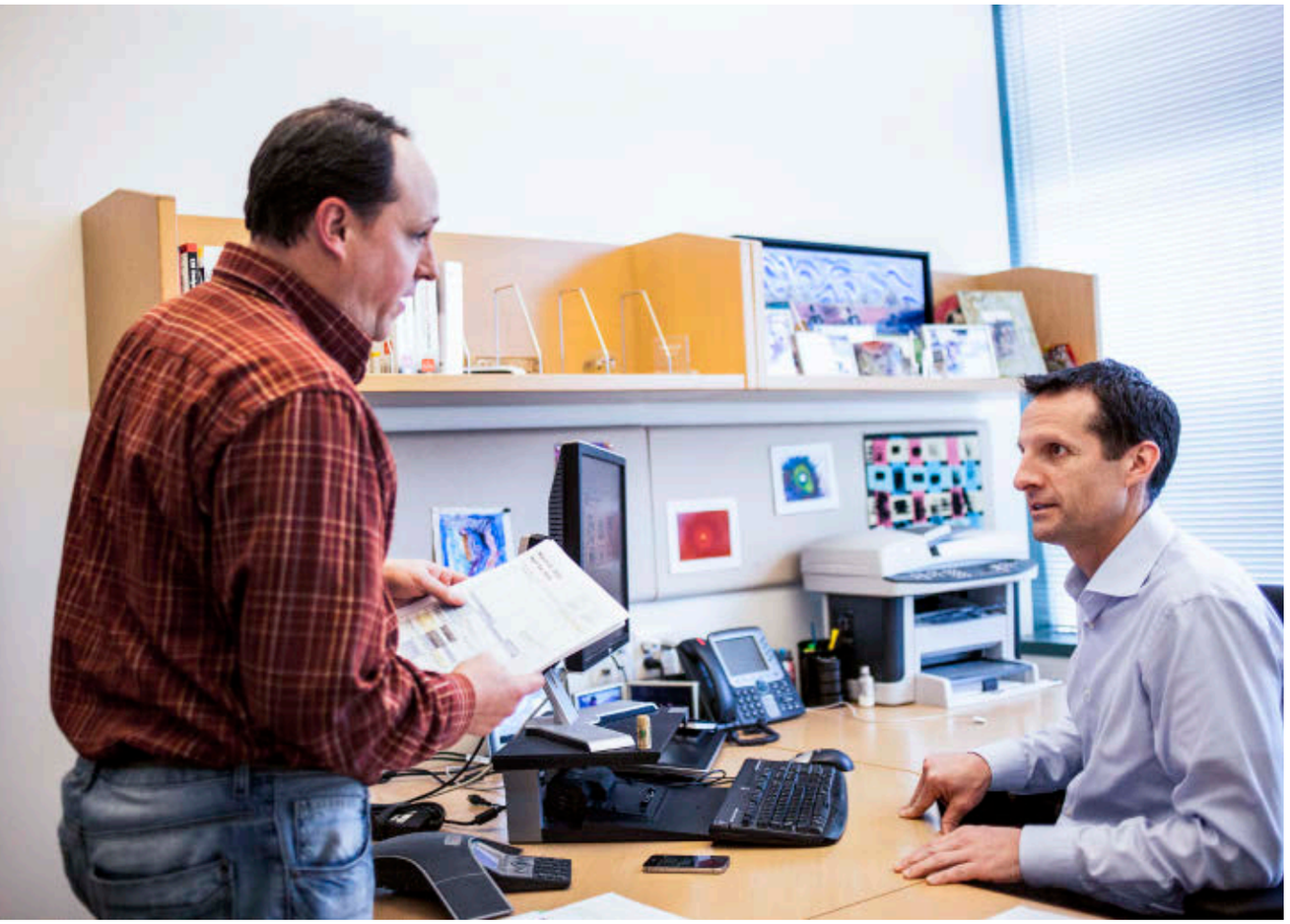
from the design phase throughout the lifecycle.

These changes require an evolution of the tooling and infrastructure that are used to design and develop applications.

Java Magazine: What is Oracle doing to address these requirements?

Tonas: Some of the work has already happened. For example, NetBeans has supported the Java and JavaScript combination for a few releases now. Looking forward, Oracle has several new and innovative browser-based, cloud-centric, and mobile initiatives under way that we will be sharing with the community over the next several months.

We are leveraging skills and technology from across our current developer tools organization to develop these new capabilities.



Brian Fry, director of product management, chats with Tonas about Oracle's cloud tooling strategy.

We see the new generation of developer tools as complementary to the tools that developers use and love today. The first of these initiatives that you'll be able to use will be the forthcoming Oracle Developer Cloud Service—bringing your ALM [asset lifecycle management] and team collaboration work to the cloud. [You can read more about it.]

Java Magazine: You mentioned that enterprises are focusing on mobile development. Does Java fit into mobile development, and what is Oracle's solution there?

Tonas: Java is key to mobile development at Oracle, both for on-device apps and mobile middleware. Oracle Mobile Platform is designed to be modular so you can use individual components or the entire platform. We're in the process of expanding our mobile platform, but it's already an impressive offering.

We're focusing on making Java developers successful in mobile development with Oracle Mobile Application Framework. Developers can write code in Java and run it on iOS or Android tablets and phones. Developers can also write JavaScript/HTML5 or a combination of Java and JavaScript/HTML5 and deploy it in the same manner.

Oracle Mobile Suite contains everything an enterprise needs to mobilize its on-premises and cloud-deployed applications, and all components are based on Java. Integration to these systems is exposed as REST APIs for easy consumption in the mobile application framework.

Oracle Mobile Security Suite ensures that enterprises can safely provide access to sensitive enterprise applications and data on users' own mobile devices, and it integrates with existing identity and access solutions in the enterprise.

Java Magazine: Where can developers learn more about these new tools?

Tonas: Just as every year, Oracle's full vision for the future of software development will be shared at JavaOne and Oracle OpenWorld later this year. Our team is looking forward to sharing what we are working on with the development community. `</article>`

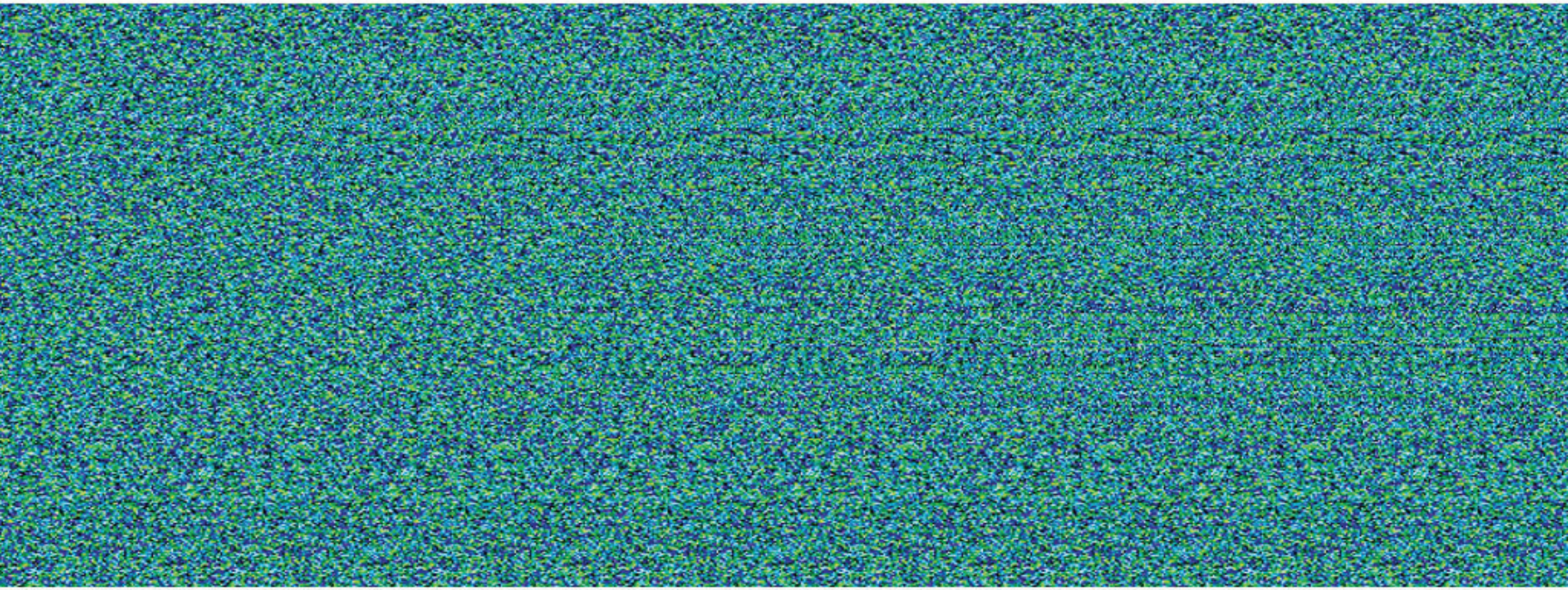
MORE ON TOPIC:

As Senior Community Manager for Java Developers for Oracle Technology Network, **Tori Wieldt** manages the Java developer experience across all channels of Oracle's developer program: OTN/Java, @Java, *Java Magazine*, blogs.oracle.com/java, and YouTube/Java. She loves to travel and to interview developers around the world. She has been involved in technology as a tech writer, sys admin, web manager, and marketeer.

LEARN MORE

- NetBeans IDE
- Oracle JDeveloper
- Oracle Enterprise Pack for Eclipse

The answer is right in front of you



Java Image Enabling SDKs that Help You See the Big Picture

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

Click for tips on viewing the stereogram

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

www.atalasoft.com

Quick and Easy Conversion to Java SE 8 with NetBeans IDE 8

New tools in NetBeans 8 for leveraging the functional features of Java SE 8

GEERTJAN WIELENGA, **LYLE** FRANKLIN, AND **ALEX** GYORI

BIO

In this article, we're going to look at new and helpful tools in NetBeans IDE 8 that enable developers to use the functional features of Java SE 8. Java SE 8 introduces lambda expressions to the language. **Figure 1** shows a lambda expression in Java. The parameters are shown on the left side of the lambda arrow, while the method body is shown to its right.

NetBeans IDE 8 provides two automated refactorings that leverage this new language feature. The first refactoring converts anonymous inner classes to lambda expressions, and the second refactoring converts for loops that iterate over collections to functional operations that use lambda expressions.

We'll start by looking at the first refactoring. Anonymous inner classes such as the one shown in **Figure 2** are useful constructs, although the syntax is unnecessarily verbose. Lambda expressions provide a more concise replacement for anonymous inner classes that declare only a single method.

NetBeans IDE 8 highlights valid refactoring sites (see **Figure 3**), and when you select the Java hint in the left sidebar, the IDE performs the conversion automatically. As

```
ActionListener listener = (ActionEvent e) -> {
    System.out.println("hello lambda"); };
```

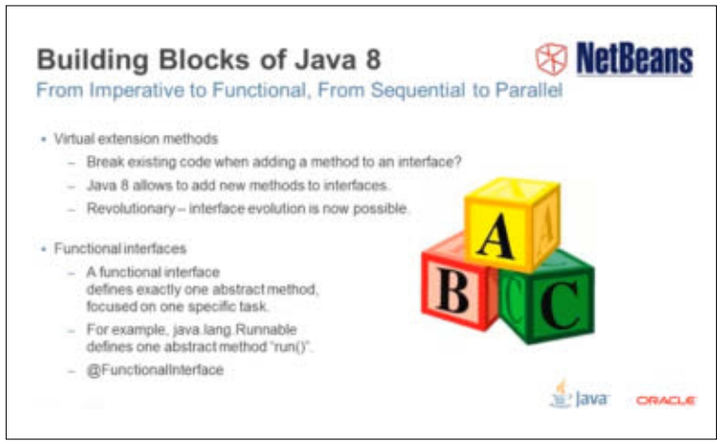
Figure 1

```
21  JButton testButton = new JButton("Test Button");
    testButton.addActionListener(new ActionListener() {
23      @Override
        public void actionPerformed(ActionEvent ae) {
25          System.out.println("Click Detected by Anon Class");
26      }
27  });
```

Figure 2

```
21  JButton testButton = new JButton("Test Button");
    testButton.addActionListener(new ActionListener() {
23      Use lambda expression
        public void actionPerformed(ActionEvent ae) {
25          System.out.println("Click Detected by Anon Class");
26      }
27  });
```

Figure 3



Development Tools and Techniques

Geertjan Wielenga shows you how to migrate your code to Java SE 8.

you can see in **Figure 4**, the new syntax is much more concise and easier to read.

While this transformation might seem simple at first glance, the IDE performs static analysis to ensure that the transformation is performed safely. For example, examine the code snippet in **Figure 5**.

In this method, the doPrivileged method is overloaded and can accept either a PrivilegedAction or a PrivilegedExceptionAction. When you perform the refactoring, the IDE automatically adds a cast to the correct type (see **Figure 6**).

If this cast were not added, the resulting lambda expression would be ambiguous, as shown in **Figure 7**, and a compilation error would result. In **Figure 8**, a variable declaration in the anonymous inner class hides or shadows a variable declared in the enclosing scope.

While variable shadowing is legal in anonymous inner classes, lambda expressions do not allow this. The IDE detects variable shadowing and automatically renames shadowing variables to avoid compilation errors. As you can see in **Figure 9**, a 1 was appended to the variable name to avoid any errors.

The second refactoring, which is shown in **Figure 10**, highlights another feature of Java SE 8: internal iterators. Java SE 8 has added several methods to the Collections API that take a lambda expression as an argument and enable functional operations over collections. Some examples of these operations are MapReduce, filterForEach, and anyMatch, to name a few. The IDE

```
21  JButton testButton = new JButton("Test Button");
22  testButton.addActionListener((ActionEvent ae) -> {
23      System.out.println("Click Detected by Anon Class");
24  });
```

Figure 4

```
27  try {
28      conv = AccessController.doPrivileged(
                new PrivilegedExceptionAction<B2CConverter>() {
30              @Override
                public B2CConverter run() throws Exception {
32                  return new B2CConverter(enc);
33              }
34          });
35  } catch (PrivilegedActionException ex) {
36      Exception e = ex.getException();
37      if (e instanceof IOException) {
38          throw (IOException) e;
39      }
40  }
```

Figure 5

```
27  try {
28      conv = AccessController.doPrivileged(
29              (PrivilegedExceptionAction<B2CConverter>) () -> new B2C
30  } catch (PrivilegedActionException ex) {
31      Exception e = ex.getException();
32      if (e instanceof IOException) {
33          throw (IOException) e;
34      }
35  }
```

Figure 6

```
27  try {
        conv = AccessController.doPrivileged(
29              () -> new B2CConverter(enc));
30  } catch (PrivilegedActionException ex) {
31      Exception e = ex.getException();
32      if (e instanceof IOException) {
33          throw (IOException) e;
34      }
35  }
```

Figure 7

```
19  private JSlider createZoomSlider() {
20      JSlider slider = new JSlider();
21      slider.setFocusable(true);
22      slider.setMinimum(1);
23      slider.setMaximum(800);
24      slider.setValue(100);
        slider.addChangeListener(new ChangeListener() {
26          @Override
            public void stateChanged(ChangeEvent event) {
28              JSlider slider = (JSlider)event.getSource();
29
30          }
31      });
32      return slider;
33  }
```

Figure 8

can refactor existing external iterators, such as enhanced for loops, to utilize these new internal iterators.

In the example in **Figure 11**, the total number of errors in a collection of rules is calculated. Although it might not be obvious, this example is analogous to a filter MapReduce operation. The if statement filters out elements, the function call maps a rule to the number of errors, and the compound assignment adds each count to an accumulator. The IDE can automatically refactor this code to use functional operations, as shown in **Figure 12**.

This example also shows that the types of the parameters of the lambda expressions can be omitted and can instead be inferred by the compiler. However, you might find it useful for readability or style reasons to let the IDE create the parameter types for you (see **Figure 13**).

As you can see in **Figure 14**, in line 10, the parameter type has now been set explicitly.

Conversion to method references—the alternative way of working with lambda expressions—

```
19  private JSlider createZoomSlider() {
20      JSlider slider = new JSlider();
21      slider.setFocusable(true);
22      slider.setMinimum(1);
23      slider.setMaximum(800);
24      slider.setValue(100);
25      slider.addChangeListener((ChangeEvent event) -> {
26          JSlider slider1 = (JSlider)event.getSource();
27      });
28      return slider;
29  }
```

Figure 9

```
public int getTotalWeight(List<Block> blocks) {
    return blocks.stream()
            .map(b -> b.getWeight())
            .reduce(0, (a, b) -> a + b);
}

public void changeColor(List<Block> blocks) {
    blocks.stream()
            .filter(b -> b.getColor() == Color.RED)
            .forEach(b -> b.setColor(Color.BLUE));
}

public boolean areAnyBlocksBlue(List<Block> blocks){
    return blocks.stream().anyMatch(b -> b.getColor() == Color.BLUE);
}
```

Figure 10

```
7   public int getNumberOfErrors(String grammarName){
8       int count = 0;
        for (ElementRule rule : getRules()) {
10          if(rule.hasErrors()){
11              count += rule.getErrorCount();
12          }
13      }
14      return count;
15  }
```

Figure 11

```
7   public int getNumberOfErrors(String grammarName) {
8       int count = 0;
9       count = getRules().stream().
                filter((rule) -> (rule.hasErrors())).
10              map((rule) -> rule.getErrorCount()).
11              reduce(count, Integer::sum);
12      return count;
13  }
14
```

Figure 12

```
9       count = getRules().stream().
                filter((rule) -> (rule.hasErrors())).
11              .getErrorCount()).
12              ger::sum);
13
14  }
```

Use explicit parameter types
Use anonymous innerclass
Use block as the lambda's body

Figure 13

```
9       count = getRules().stream().
                filter((ElementRule rule) -> (rule.hasErrors())).
11              map((rule) -> rule.getErrorCount()).
12              reduce(count, Integer::sum);
13      return count;
14  }
```

Figure 14

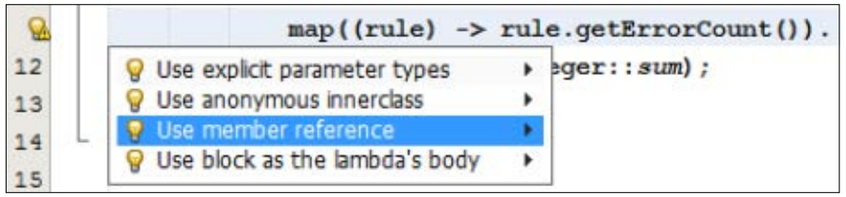


Figure 15

```
count = getRules().stream().
        filter((rule) -> (rule.hasErrors()))
        map(ElementRule::getErrorCount).
        reduce(count, Integer::sum);
return count;
```

Figure 16

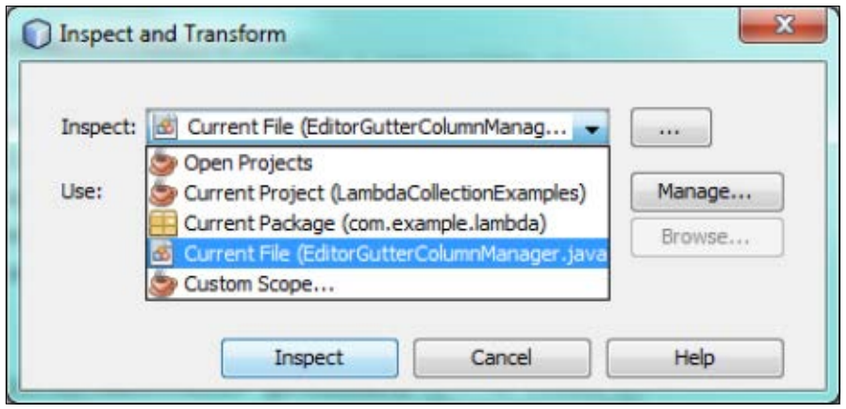


Figure 17

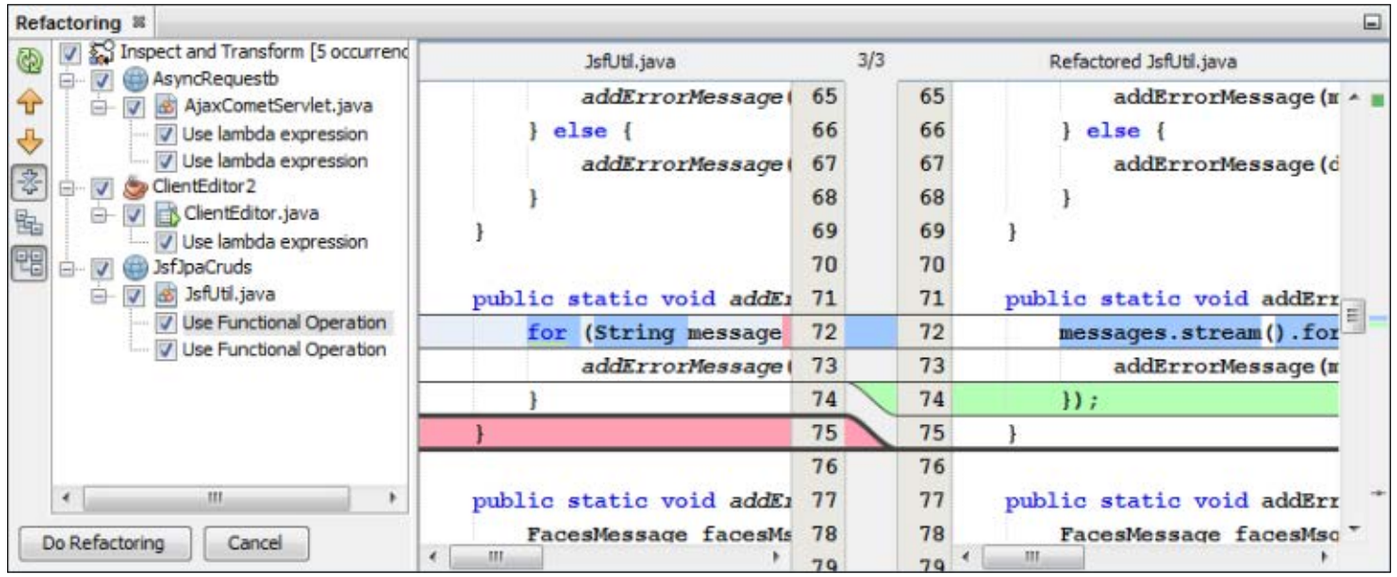


Figure 18

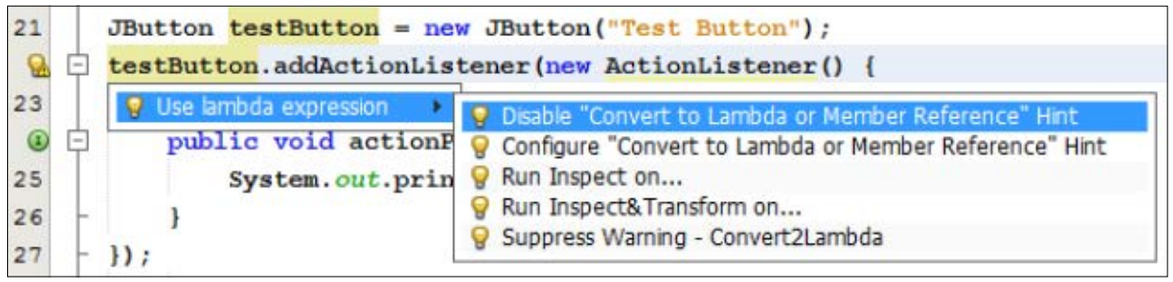


Figure 19

is also supported by the IDE (see **Figure 15**). As shown in **Figure 16**, the :: syntax has now replaced the -> syntax of standard lambda expressions.

Without the aid of an automated tool, these refactorings are time consuming and can introduce subtle bugs if done incorrectly. NetBeans IDE performs these refactorings quickly, while preserving the original program behavior.

In addition to the quick hint mode of operation, the IDE can function in batch mode. As shown in **Figure 17**, you can select a single file, an entire project, or all open projects, and the IDE will generate a preview of all possible conversions for that selection.

Instead of using the batch refactoring tool, whenever you see a Java hint in the left sidebar you can specify that the refactoring should be done across a scope of your choice (see **Figure 18**).

Even projects with hundreds of thousands of lines of code can be scanned in a matter of seconds. The preview allows for fine-grained control over which conversion should take place, or it can simply refactor all possible conversions (see **Figure 19**).

Conclusion

NetBeans IDE 8 provides the tools you need to smoothly and correctly upgrade your imperative code constructs to the new functional style introduced in Java SE 8. Its automated refactoring capability can refactor your code to use functional operations, and its batch mode allows you to specify the scope of the refactoring. Get started converting your code today! `</article>`

MORE ON TOPIC:

LEARN MORE

- Download NetBeans IDE 8

BEN EVANS

BIO

Exploring Java 8 Profiles

What can Compact Profiles do for your application?

Java 8 introduces the concept of Compact Profiles, which are reduced versions of the Java runtime environment (JRE) that do not contain the usual full contents of rt.jar. In this article, we will explore the advantages of using Compact Profiles and how they point the way toward a modular future for the JDK.

The current versions of the JRE are quite monolithic: rt.jar is at least 60 MB on its own, without taking into account the size of native code loaded as dynamic libraries. If we can reduce the size of the Java platform footprint and move to a modular view of the JDK, we can realize some great benefits:

- Faster Java Virtual Machine (JVM) startup times
- Reduced resource consumption
- Removal of packages that, in hindsight, shouldn't be in the core
- Improved security, because removing unused classes reduces the attack surface of the platform
- Convergence of the Java ME Connected Device Configuration (CDC) with Java SE

The initial approach to this effort was a full modularization of the platform, known as Project Jigsaw. This ambitious project also included other goals:

- Incorporate best practices for dependencies into the platform core, and apply lessons learned about dependency management from tools such as Maven, Apache Ivy, OSGi standard, and Linux distributions.
- Isolate dependencies—solve the library versioning problem.
- Allow application developers to package their code as modules, rather than as JAR files.

The catch is that in order to achieve the full set of goals, major surgery on the Java platform core is required. In particular, a new approach to classloading—involving a "modular classloader"—is required. This has a lot of edge cases and is a complex undertaking, especially if we need to maintain backward compatibility.

In order to maintain the release date for Java 8, a decision was made to move full modularity out to Java 9, targeted for 2016. Java 8 Compact Profiles are designed to be a first step toward full modularity, with some of the basic advantages noted above. They build on the initial work done for modularity and provide a cut-down Java runtime, which

- Is fully compliant with the JVM and Java language specifications
- Has a substantially reduced footprint
- Removes functionality that is not always needed (for example, CORBA)
- Works for many applications (especially server-side code)

Compact Profiles are based on packages; they contain a number of full packages, and no partial packages are currently allowed. They are also subject to two other restrictions:

- A profile must form a closed set; references to classes not contained in the profile are not allowed.
- If a profile contains some classes from another, smaller profile, it must contain all of them, so partially overlapping profiles are not allowed. Put another way, profiles are additive.

The smallest Compact Profile is called compact1, and it comprises the following packages: java.io, java.lang, java.lang.annotation, java.lang.invoke, java.lang.ref, java.lang.reflect, java.math, java.net, java.nio, java.nio.channels,

PHOTOGRAPH BY JOHN BLYTHE

java.nio.channels.spi, java.nio.charset, java.nio.charset.spi, java.nio.file, java.nio.file.attribute, java.nio.file.spi, java.security, java.security.cert, java.security.interfaces, java.security.spec, java.text, java.text.spi, java.time, java.time.chrono, java.time.format, java.time.temporal, java.time.zone, java.util, java.util.concurrent, java.util.concurrent.atomic, java.util.concurrent.locks, java.util.function, java.util.jar, java.util.logging, java.util.regex, java.util.spi, java.util.stream, java.util.zip, javax.crypto, javax.crypto.interfaces, javax.crypto.spec, javax.net, javax.net.ssl, javax.script, javax.security.auth, javax.security.auth.callback, javax.security.auth.login, javax.security.auth.spi, javax.security.auth.x500, and javax.security.cert.

Two other Compact Profiles are currently specified: compact2, which adds packages used for remote method invocation (RMI), SQL, and XML, and compact3, which comprises all of compact2 plus tooling and management packages (including Java Management Extensions [JMX]) as well as additional cryptography libraries. The smallest profile, compact1, occupies around 11 MB, which is a significant space savings.

Note that as currently specified, all of these profiles are headless; they do not contain any GUI classes. Any applications requiring GUI support (Swing or AWT) must use a full JRE.

GET SMALLER

Java 8 introduces the concept of Compact Profiles, which are reduced versions of the Java runtime environment (JRE) that do not contain the usual full contents of rt.jar.

Tools

To make use of Compact Profiles, developers require tools. One important question is whether an application can run against a specific profile. Java 8 ships with two tools that have been enhanced to help answer this question: both javac and jdeps have been modified to be aware of profiles.

javac is the tool of choice for determining whether a collection of source code can be safely run on a specific profile. This is achieved by using the new -profile switch. javac -profile <profile> will cause a compilation error to be generated for any usage of a class not present in the indicated profile.

In some cases, however, source code is not available or a recompilation run is inconvenient.

LISTING 1 LISTING 2 LISTING 3

```
jdeps -s -P junit.jar
junit.jar              -> compact1
```

Download all listings in this issue as text

Fortunately, in this case, the new jdeps tool can help.

jdeps is a new static analysis tool that ships with Java 8. It provides an analysis of the dependencies that a specific class or JAR file has. This tool is extremely useful (and not just for profile dependencies) and features a -profile (or -P) switch that indicates which packages depend on which profiles.

Let's take a look at an example, which summarizes the package dependencies for the popular JUnit testing library. See **Listing 1**, which shows good news; everyone should be able to test their code.

If we want more information, we can use the -v switch for verbose output, which will give us a lot of detail about each class inside the JAR file. See **Listing 2** (some of the output was truncated because it was 2,152 lines long).

If we want a slightly more high-level view, we can use -V package to show dependencies between packages, as shown in **Listing 3** (some of the output was truncated because it was 1,297 lines long).

jdeps is also very flexible about what it will accept as input: a JAR file, a directory, or a single .class file. It provides capabilities for recursive

traversal and for specifying that only packages with a name that matches a given regular expression should be considered. It can also warn that code uses an internal API and is not portable between Java environments (and might break if run against a future version of Java).

Finally, let's look at the NetBeans IDE. The current version is 8.0, which already has support for a wide range of JDK 8 features, including Compact Profiles. When selecting which JDK or JRE to use in Project Properties, for JDK 8 and later, a developer can choose whether to compile against the full JRE or a profile. This makes it much easier to ensure that when targeting a particular profile, unwanted dependencies don't creep in. With luck, other IDEs will follow suit and also add support to allow developers to write code in the IDE that checks conformance with a specific profile at development time.

YOU CHOOSE
A developer can choose whether to compile against the full JRE or a profile.

A Word About Stripped Implementations

In addition to Compact Profiles, there is another proposed new technique for reducing resource utilization for deployed JVM applications: Stripped Implementations. A Stripped Implementation is a reduced JRE, which is packaged with an application that has the exclusive use of it. Because the application is the only possible client for the Stripped Implementation, the runtime can be aggressively pruned, removing packages, classes, and even methods that are not used by the application.

This approach is advantageous in circumstances where resource limitations are severe. It relies on extensive testing to ensure that nothing that the application could rely on is removed by the stripping process. In general, it is extremely difficult to get a precise accounting of an application's dependencies. This is due in large part to the existence of techniques such as reflection and classloading, which can greatly complicate (or even render impossible) the task of ascertaining the true dependencies of a set of classes.

Compact Profiles are very different from Stripped Implementations—the former are Java runtimes designed for general-purpose use and are complete implementations of the Java language specification, whereas Stripped Implementations are defined to be single-use, nonreusable implementations that do not have to conform to the Java language specification at their point of delivery to the user.

Stripped Implementations were targeted as a feature for Java SE 8, but due to some complications with licensing and the current testing kit, they had to be dropped very close to the release date. Nevertheless, the intention is for Stripped Implementations to become part of the platform shortly after the release of Java 8, and when combined with Compact Profiles, they have the potential to provide extremely small footprints for single-use applications.

Conclusion

Java 8 Compact Profiles represent a significant step toward future goals for the platform—both in terms of embedded (or capability-restricted) development and also for server-side developers. While not being the complete modularity solution we might have wished for in Java 8, Compact Profiles are a useful development in their own right. `</article>`

LEARN MORE

- Compact Profiles Demonstrated
- Compact Profiles Overview
- Hinkmond Wong's EclipseCon 2014 presentation on Compact Profiles

CREATE THE FUTURE
oracle.com/java
Java™
ORACLE

RAOUL-GABRIEL URMA

BIO

Part 2

Processing Data with Java SE 8 Streams

Combine advanced operations of the Stream API to express rich data processing queries.

In the first part of this series, you saw that streams let you process collections with database-like operations. As a refresher, the example in **Listing 1** shows how to sum the values of only expensive transactions using the Stream API. We set up a pipeline of operations consisting of intermediate operations (filter, map) and a terminal operation (reduce), as illustrated in **Figure 1**.

However, the first part of this series didn't investigate two operations:

- flatMap: An intermediate operation that lets you combine a "map" and a "flatten" operation
- collect: A terminal operation that takes as an argument various recipes (called *collectors*) for accumulating the elements of a stream into a summary result

These two operations are useful for expressing more-complex queries. For instance, you can combine flatMap and collect to produce a Map representing the number of occurrences of each character that appears in a stream of words, as shown in **Listing 2**. Don't worry if this code seems overwhelming at first. The purpose of this article is to explain and explore these two operations in more detail.

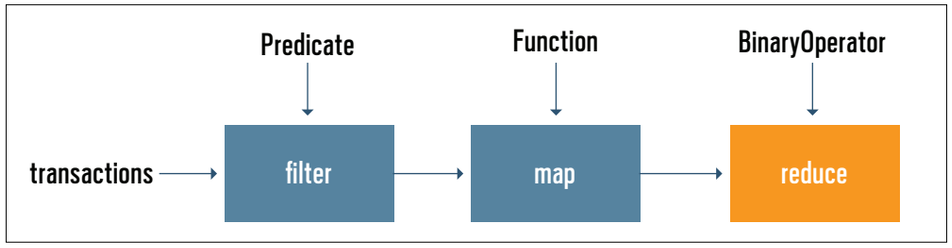


Figure 1

The code in **Listing 2** will produce the output shown in **Listing 3**. Awesome, isn't it? Let's get started and explore how the flatMap and collect operations work.

flatMap Operation

Suppose you would like to find all unique words in a file. How would you go about it?

You might think that it is easy; we can use Files.lines(), which you saw in the previous article, because it can return a stream consisting of the lines of a file. We can then split each line into words using a map() operation and, finally, use the operation distinct() to remove duplicates. A first attempt could be the code shown in **Listing 4**.

Unfortunately, this is not quite right. If you run this code, you will get puzzling output similar to this:

```
[Ljava.lang.String;@7cca494b
[Ljava.lang.String;@7ba4f24f
...
```

Our first attempt is actually printing the String representation of several streams! What is happening? The problem with this approach is that the lambda passed to the map method returns an array of String (String[]) for each line in the file. As a result, the stream returned by the map method is actually of type Stream<String[]>. What we really want is Stream<String> to represent a stream of words.

Luckily there's a solution to this problem using the method flatMap. Let's see

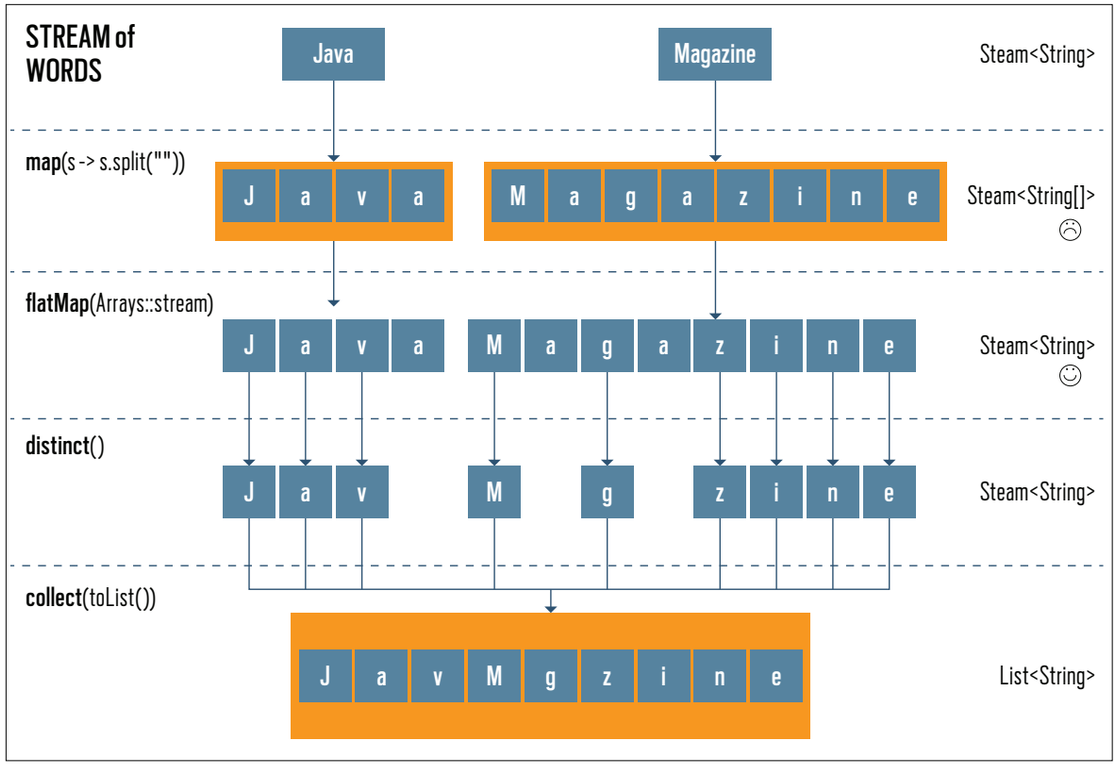


Figure 2

step-by-step how to get to the right solution.

First, we need a stream of words instead of a stream of arrays. There's a method called Arrays.stream() that takes an array and produces a stream. See the example shown in **Listing 5**.

Let's use it in our previous stream pipeline to see what happens (see **Listing 6**). The solution still doesn't work. This is because we now end up with a list of streams of streams (more precisely a Stream<Stream<String>>). Indeed, we first convert each line into an array of words, and then convert each array into a separate stream using the method Arrays.stream().

We can fix this problem by using a flatMap, as shown in **Listing 7**. Using the flatMap method has the effect of replacing each generated array not by a stream but by the contents of that stream. In other words, all the separate streams that were generated when using map(Arrays::stream) get amalgamated or "flattened" into one single stream. **Figure 2** illustrates the effect of using the flatMap method.

In a nutshell, flatMap lets you replace each value of a stream with another stream, and then it concatenates all the generated streams into one single stream.

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

```
int sumExpensive =
    transactions.stream()
                        .filter(t -> t.getValue() > 1000)
                        .map(Transaction::getValue)
                        .reduce(0, Integer::sum);
```

Download all listings in this issue as text

Note that flatMap is a common pattern. You will see it again when dealing with Optional or CompletableFuture.

collect Operation

Let's now look at the collect method in more detail. The operations you saw in the first part of this series were either returning another stream (that is, they were intermediate operations) or returning a value, such as a boolean, an int, or an Optional object (that is, they were terminal operations).

The collect method is a terminal operation, but it is a bit different because you used it to transform a stream into a list. For example, to get a list of the IDs for all the expensive transactions, you can use the code shown in **Listing 8**.

The argument passed to collect is an object of type java.util.stream.Collector. What does a Collector object do? It essentially describes a recipe for accumulating the elements of a stream into a final result. The factory method Collectors.toList() used earlier returns a Collector describing how to accumulate a stream into a list. However, there are many similar built-in Collectors available.

Collecting a stream into other collections. For example, using toSet() you can convert a stream into a Set, which will remove duplicate elements. The code in **Listing 9** shows how to generate the set of only the cities that have expensive transactions. (Note: In all future examples, we assume that the factory methods of the Collectors class are statically imported using import static java.util.stream.Collectors.*.)

Note that there are no guarantees about what type of Set is returned.

However, using toCollection() you can have more control. For example, you can ask for a HashSet by passing a constructor reference to it (see **Listing 10**).

However, that's not all you can do with collect and collectors. It is actually a very tiny part of what you can do. Here are some examples of what you can express:

- Grouping a list of transactions by currency to the sum of the values of all transactions with that currency (returning a Map<Currency, Integer>)
- Partitioning a list of transactions into two groups: expensive and not expensive (returning a Map<Boolean, List<Transaction>>)
- Creating multilevel groupings, such as grouping transactions by cities and then further categorizing by whether they are expensive or not (returning a Map<String, Map<Boolean, List<Transaction>>>)

Excited? Great. Let's see how you can express these queries using the Stream API and collectors. We first start with a simple example that "summarizes" a stream: calculating the average, the maximum, and the minimum of a stream. We then look at how to express simple groupings, and finally we see how to combine collectors together to create powerful queries, such as multilevel groupings.

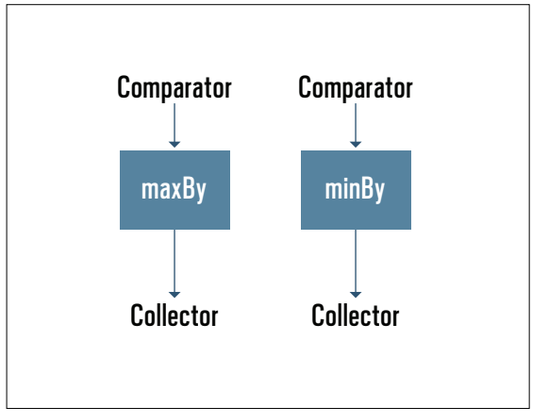


Figure 3

Summarizing. Let's warm up with some simple examples. You saw in the previous article how to calculate the number of elements, the maximum, the minimum, and the average of a stream using the reduce operation and using primitive streams. There are predefined collectors that let you do just that as well. For example, you can use counting() to count the number of items, as shown in **Listing 11**.

You can use summingDouble(), summingInt(), and summingLong() to sum the values of a Double, an Int, or a Long property of the elements in a stream. In **Listing 12**, we calculate the total value of all transactions.

Similarly, you can use averagingDouble(), averagingInt(), and averagingLong() to calculate the average, as shown in **Listing 13**.

In addition, by using maxBy() and minBy() you can calculate the maximum and minimum element of a stream. However, there's a catch: you need to define an order for the elements of a stream to be able to compare them. This is why maxBy and minBy take a Comparator object as an argument, as illustrated in **Figure 3**.

In the example in **Listing 14**, we use the static method comparing(), which generates a Comparator object from a function passed as an argument. The function is used to extract a comparable key from the element of a stream. In this case, we find the highest transaction by using the value of a transaction as a comparison key.

There's also a reducing() collector that lets you combine all elements in a stream by repetitively applying an operation until a result is produced. It is conceptually similar to the reduce() method you saw previously. For example, **Listing 15** shows an alternative way to calculate the sum of all transactions using reducing().

reducing() takes three arguments:

- An initial value (it is returned if the stream is empty); in this case, it is 0.
- A function to apply to each element of a stream; in this case, we extract the value of each transaction.
- An operation to combine two values produced by the extracting function; in this case, we just add up the values.

You might say, "Wait a minute; I can already do that with other stream methods, such as reduce(), max(), and min(), so why are you showing me this?" You will see later that we can combine collectors to

QUERY POWER

Using the Stream API and collectors, you can combine collectors together to create powerful queries, such as multilevel groupings.

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

```
Files.lines(Paths.get("stuff.txt"))
        .map(line -> line.split("\\s+")) // Stream<String[]>
        .flatMap(Arrays::stream) // Stream<String>
        .distinct() // Stream<String>
        .forEach(System.out::println);
```

Download all listings in this issue as text

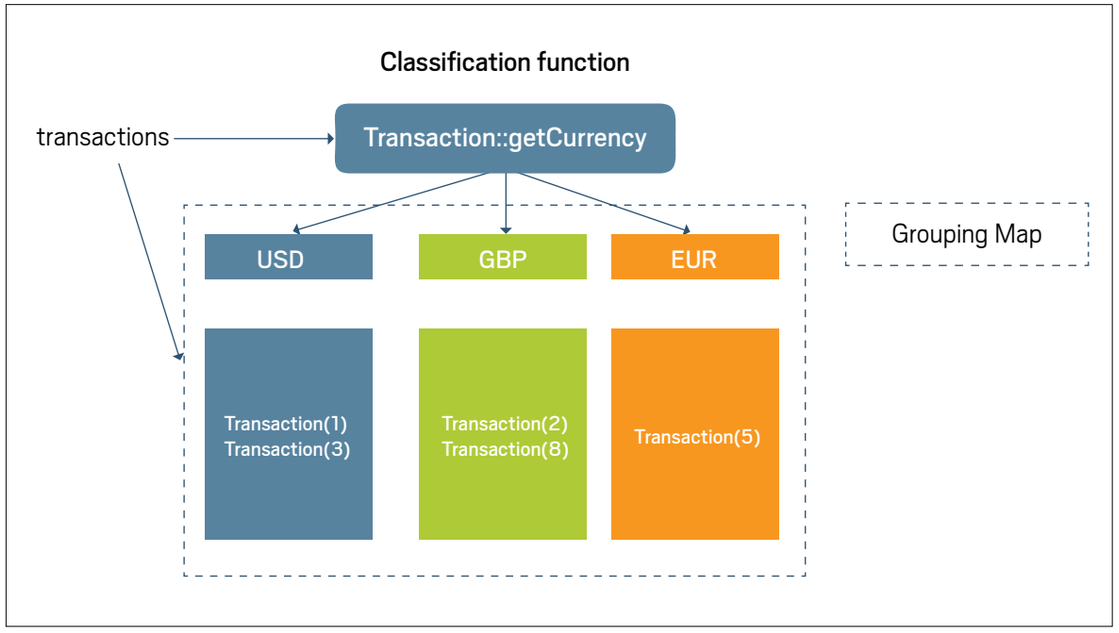


Figure 4

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

```
double average = transactions.stream().collect(
        averagingInt(Transaction::getValue));
```

Download all listings in this issue as text

build more-complex queries (for example, grouping plus averaging), so it is handy to know about these built-in collectors as well.

Grouping. A common database query is to group data using a property. For example, you might want to group a list of transactions by currency. Expressing such a query using explicit iteration is somewhat painful, as you can see in the code in **Listing 16**.

You need to first create a Map where the transactions will be accumulated. Then you need to iterate the list of transactions and extract the currency for each transaction. Before adding the transaction in as a value in the Map, you need to check whether a list has been created, and so on. It is a shame, because fundamentally all we want to do is "group the transactions by currency." Why does it have to involve so much code? Good news: there's a collector called groupingBy() that lets us express such use cases in a concise way. We can express the same query as shown in **Listing 17**, and now the code reads closer to the problem statement.

The groupingBy() factory method takes as an argument a function for extracting the key used to classify the transactions. We call it a *classification function*. In this case, we pass a method reference, Transaction::getCurrency, to group the transaction by currency. **Figure 4** illustrates the grouping operation.

Partitioning. There's another factory method called partitioningBy() that can be viewed as a special case of groupingBy(). It takes a predicate as an argument (that is, a function that returns a boolean) and groups the elements of a stream according to whether or not they match that predicate. In other words, partitioning a stream of transactions organizes the transactions into a Map<Boolean, List<Transaction>>. For example, if you want to group the transactions into two lists—cheap and expensive—you could use the partitioningBy collector as shown in **Listing 18**, where the lambda t -> t.getValue() > 1000 is a predicate for classifying cheap and expensive transactions.

Composing collectors. If you are familiar with SQL, you might know that you can combine GROUP BY with functions such as COUNT() and SUM() to group transactions by currency and by their sum. So, can we do something similar with the Stream API? Yes. Actually, there's an overloaded version of groupingBy() that takes another collector object as a second argument. This additional collector is used to define how to accumulate all the elements that were associated with a key using the groupingBy collector.

OK, this sounds a bit abstract, so let's look at a simple example. We would like to generate a Map of cities according to the sum of all transactions for each city (see

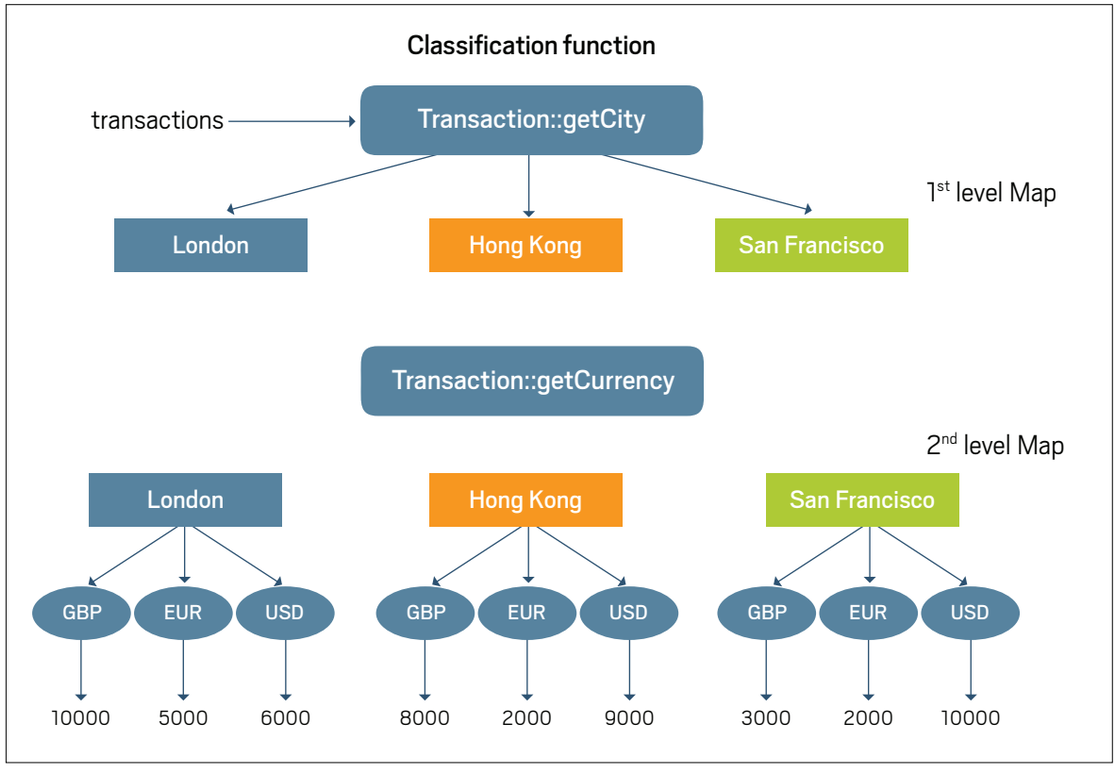


Figure 5

Listing 19). Here, we tell groupingBy to use the method getCity() as a classification function. As a result, the keys of the resulting Map will be cities. We would normally expect a List<Transaction> back as the value for each key of the Map using the basic groupingBy.

However, we are passing an additional collector, summingInt(), which sums all the values of the transactions associated with a city. As a result, we get a Map<String, Integer> that maps each city with the total value of all transactions for that city. Cool, isn't it? Think about it: the basic version of groupingBy (Transaction::getCity) is actually just shorthand for writing groupingBy (Transaction::getCity, toList()).

Let's look at another example. How about if you want to generate a Map of the highest-valued transaction for each city? You might have guessed that we can reuse the maxBy collector we defined earlier, as shown in **Listing 20**.

You can see that the Stream API is really expressive; we are now building some really interesting queries that can be written concisely. Can you imagine going back to processing a collection iteratively?

Let's look at a more-complicated example to finish. You saw that groupingBy can take another collector object as an argument to accumulate the elements according to a further classification. Because groupingBy is a collector itself, we can create multilevel groupings by passing another groupingBy collector that defines a second criterion by which to classify the stream's elements.

LISTING 19 LISTING 20 / LISTING 21

```
Map<String, Integer> cityToSum =
      transactions.stream().collect(groupingBy(
       Transaction::getCity, summingInt(Transaction::getValue)));
```

Download all listings in this issue as text

In the code in **Listing 21**, we group the transactions by city, and then we further group the transactions by the currency of transactions in each city to get the average transaction value for that currency. **Figure 5** illustrates the mechanism.

Creating your own collector. All the collectors we showed so far implement the interface java.util.stream.Collector. This means that you can implement your own collectors to define "customized" reduction operations. However, this subject could easily fit into another article, so we won't discuss it here.

Conclusion

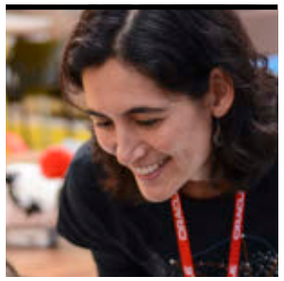
In this article, we've explored two advanced operations of the Stream API: flatMap and collect. They are tools that you can add to your arsenal for expressing rich data processing queries.

In particular, you've seen that the collect method can be used to express summarizing, grouping, and partitioning operations. In addition, these operations can be combined to build even richer queries, such as "produce a two-level Map of the average transaction value for each currency in each city."

However, this article didn't investigate all the built-in collectors that are available. We invite you to take a look at the Collectors class and try out other collectors, such as mapping(), joining(), and collectingAndThen(), which you might find useful. `</article>`

LEARN MORE

- *Java 8 in Action: Lambdas, Streams, and functional-style programming*



SASKIA VERMEER-OOMS, **RÉGINA** TEN BUGGENCATE, AND **LINDA** VAN DER PAL

BIO

Development Tools and Techniques

A Sprint in the Life of a Scrum Master

Saskia Vermeer-Ooms gives a tour of a Scrum sprint.

Most of us have heard of Scrum by now, but not all of us have had a chance to use it yet. And even if you have read books or taken a course, it might be difficult to decide where to start or how to apply Scrum within your own organization.

"That won't work in our organization" is a phrase you hear often, usually followed by other good reasons why (part of) Scrum will not work. Why not see what someone who has been doing Scrum for a while—someone who has managed to make Scrum work with a Scrum product owner and operations team who live and work in a different country than the rest of the team—has to say?

For those of us who haven't yet heard, Scrum *is an iterative and incremental agile software development framework for managing software projects and product or application development. According to Wikipedia, it defines "a flexible, holistic product development strategy where a development team works as a unit to reach a common goal."*

Régina ten Buggencate and Linda van der Pal sat down with Saskia Vermeer-Ooms to find out what really goes on in a Scrum development team.

ten Buggencate: Hi, Saskia. I heard you're a Scrum Master now. Can you tell us a little bit about what that's like? How about starting by telling us about your team?

Vermeer-Ooms: I am the Scrum Master of a development team that consists of three Java developers, one front-end developer, and one team member responsible for making web forms. We work for a UK-based client, and my office is based in the Netherlands. We work with a technology partner who is also based in the UK. This tech partner also does development on the same platform but, more importantly, is also responsible for the operations side of the platform. Any deployments to the live environment must go through this tech partner.

You can imagine that a lot of my time is spent e-mailing, on "chat," or in remote meetings. My main focus, however, is to facilitate my team reaching the goals we set for each sprint, which is three to four weeks long. We decided not to go for a fixed-length sprint, because most of the time a chunk of work is too large to fit within three weeks, but sometimes a four-week sprint is too long. So, instead we opted to decide on the sprint length when we plan the work for each sprint.

van der Pal: So what does a regular sprint look like? I heard it starts with a sprint planning, right?

Vermeer-Ooms: Sprint planning sessions are usually done on the first Monday of the sprint in a meeting room where we have a big whiteboard that is the same size as the Scrum board we have in our workspace. This whiteboard has plenty of different-colored Post-it notes and a screen that we project the JIRA issues on.

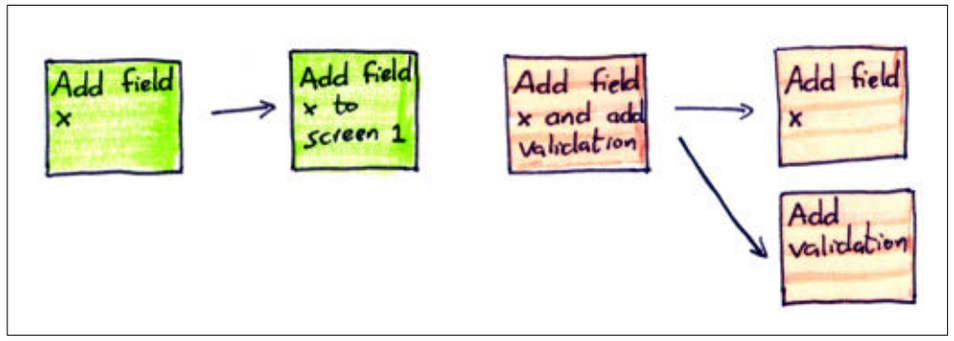


Figure 1

Before we go into the sprint planning session, I make sure that the product backlog is up to date. All issues that were not completed during the previous sprint are moved to the new sprint's fix version in JIRA. Other issues that have been prioritized by the product owner will have already been labeled under this sprint's fix version. Team members can also label for this fix version other issues they deem necessary.

I also make sure I have an overview of all the available hours for all team members. Most of my team members also work on other projects, so I use a spreadsheet for keeping track of the available hours per week for team members. I know doing time-based estimation isn't "Scrum by the books," but because of the pressure on our team members from other projects, it is a sad necessity.

At the beginning of the planning session, we put the available hours per person on one side of the whiteboard. On the other side, we put on the Post-it notes any JIRA issues that were left over from the previous sprint and any issues we have just added to the backlog.

Then we go through the issues and discuss briefly what each issue is about. If the description is not clear enough for everyone to be able to decide what is required, it is rephrased or sometimes the issue is split into subtasks or multiple new issues (see **Figure 1**). The main goal of this part of the planning session is to agree on what is being asked of us for each issue and to ensure that the description of what needs to be done is clear. If an issue is not yet on the board, we make a new Post-it note for it.

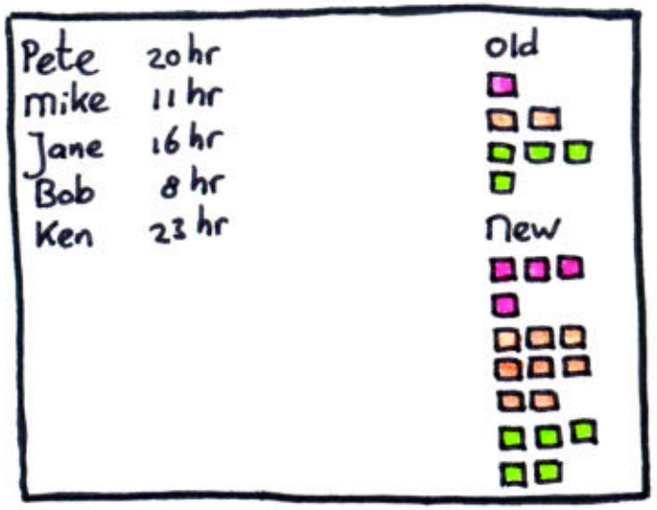


Figure 2

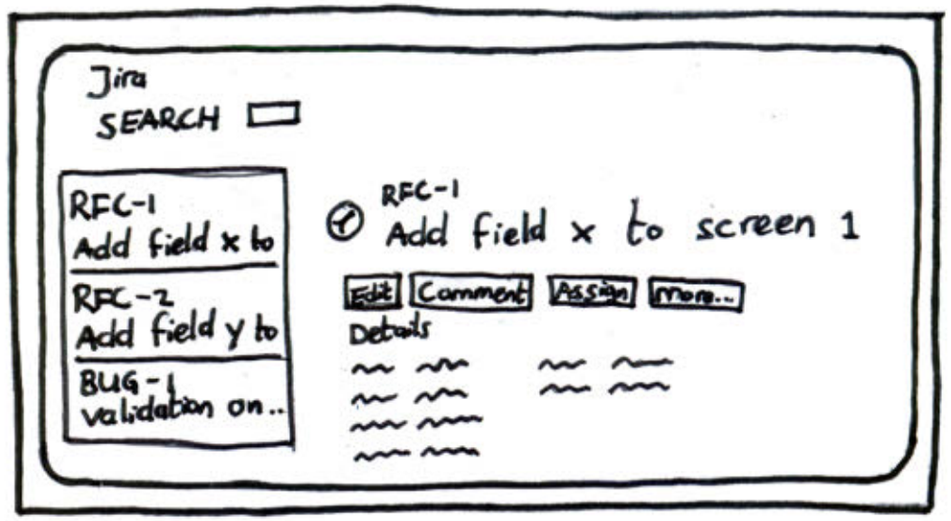


Figure 3

We have three different colors of Post-it notes (see **Figure 2**) that correspond to the three different priorities: pink for critical, orange for major, and green for minor. Each note contains the JIRA issue (see **Figure 3**), a short descriptive title, and the first initial of the team member to whom the issue is assigned.

In the second half of the planning session, we put the issues on the board by priority (highest priority on top). Then each member writes time estimates on their assigned Post-it notes. These estimates need to be agreed upon by the entire team. Everyone has their specialty in the team, but if we need to have a longer discussion about time estimates, we do. We use units of a day, with half a day being the minimum.

When we are done with this, each member adds up the hours for their own issues and writes the total next to the hours of availability that were put on the board at the start of the meeting. We usually end up with more hours than are available, which means we need to reconsider priority. Any issues that don't fit get moved to the backlog. When we all agree that the issues that are left on the board are feasible, we are done with the planning.

ten Buggencate: So you're saying that everybody estimates their own issues, but I thought that in Scrum, there was only one role for this kind of work: the Developer role.

Vermeer-Ooms: That's true, but our team members have other projects, too, and they all have their own specialties. So this is again a point where we deviate from the default laid out by the Scrum method, because this approach works better for us.

To finish our sprint planning, at the end of the meeting, I ask each team member to grab their own Post-it notes and take them to the Scrum whiteboard in our workspace, which we use for our daily standup meetings. I also make sure that JIRA is updated; the issues that are left on the board need to be taken out of this sprint version, and I can fill in the time estimates we just agreed on.

Now that the JIRA issues are clear for this sprint, I can create the draft

release notes. In these notes, I list all the issues for this sprint and also the dates that are important during this sprint. Because we have a distributed development and operations team, it is important to have these dates agreed on at the beginning of the sprint. We are dependent on our technology partner for deployment of our code to the UAT [User Acceptance Test] environment and the live environment. We have to plan precisely when we need to have our code changes ready in order to make the proposed live deployment date.

This is also the time to ask our tech partner what the current QA Git branch is. Our tech partner also does development on the same platform; therefore, it is essential that we get integration issues out of the way as soon as possible. In other words, it is essential to agree on the current QA branch so that we both have the same starting point.

van der Pal: Wow, that's quite a lot of work already. So what happens next?

Vermeer-Ooms: What follows are the normal days of work in progress. The day starts with the daily standup meeting in front of the Scrum whiteboard. Each team member informs the team what they were working on yesterday, what they will be working on today, and whether anything is blocking them from making progress.

As shown in **Figure 4**, we update the board during these standup meetings; Post-it notes are moved from the "To do" column, through the "In progress" and "To test" columns, to the "Done" column. During the daily standup meeting, we adjust any estimates that turn out to be incorrect, and we give an indication of what still needs to be done for each issue in progress.

After the standup meeting, the team gets back to work, and I do my best to resolve any impediments that were mentioned in the meeting.

ten Buggencate: So, do you have any tools to help you with quality assurance? You mentioned you were working with teams in other locations.

Vermeer-Ooms: We have a Jenkins job running a nightly build on any code committed to the current development branch. It also does an automatic deployment of our artifacts to the internal test server. This is very helpful for verifying that ongoing development is not causing other parts of the system to fail. We use this internal test server to verify work in progress and to check functionality.

Unfortunately, we do not have a dedicated tester on our team, which means that we depend on the testing capabilities of all the members of the team. That might be right according to the books, but my experience is that developers don't have the same mindset that a tester has. We try to make up for this by organizing explicit internal test sessions where the whole team tests a finished feature. We encourage each other to plan internal test sessions as much as possible. These internal test sessions are extremely valuable.

Before we start a session, we list the JIRA issues that need to be verified. We set up a shared Google doc, start testing, and write up all our findings. After the test session is over, I go through the findings and—with the help of the rest of my team—add new issues or move the issues that had made it "To test" but were not resolved back to "In progress." The issues that were verified to be resolved on our test server are labeled "Passed QA."

van der Pal: So, now that you have some work that is tested, do you also have the customer test it?

Vermeer-Ooms: Yes; by Friday, we usually have some functionality ready to show the product owner.

REALITY CHECK

I know doing time-based estimation isn't "Scrum by the books," but because of the pressure on our team members from other projects, it is a sad necessity.

Figure 4

Therefore, I need to verify with him that the new functionality we are implementing during this sprint is doing what is expected. We write up a short scenario of how we are going to use this new functionality and add a list of questions to get any uncertainties out of the way. After the product owner sends us his answers, we can carry on with the implementation.

ten Buggencate: You have three-to-four-week sprints. When do you stop building new code, and how do you hand over your code to the people who have to deploy it to production?

Vermeer-Ooms: The release date is determined by our client and tech partner; it is usually set on a Thursday. Therefore, I plan one whole week for the deployment cycle by the tech partner and set the Thursday before the release as the code cutoff for our internal integration environment.

This Thursday is the day that we want to have all our code changes in, so they can be run through our integration server, which is a different server from our development test server.

This integration server is running code that is based on the agreed-upon QA branch. Developers can commit and push code to the development branch until lunchtime. After that, the development code is merged with the QA branch and a build is started for deployment on the integration server. Other teams in our office also work on the same platform, so all parallel development comes together on the QA server at this point.

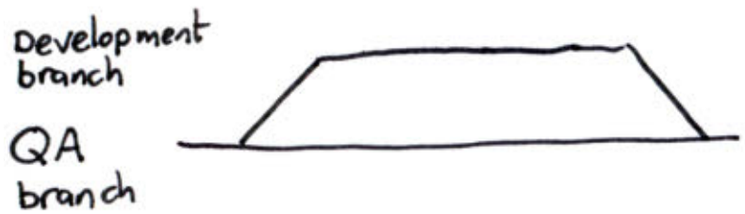


Figure 5

Release branches are named after Pokémon. I hang a printout of the current release branch on our bulletin board (see **Figure 5**).

We use the Friday before the deployment week for testing on the integration environment and, of course, for fixing any bugs that are found. We can also use this day to do demos of the newly added functionality. These demos are for each other and anyone else from the company who is interested. This is also the day to do a demo with the product owner and verify that we have implemented the functionality correctly. Because the product owner is based in the UK, we have to do his demos remotely. Therefore, sometimes his demo is separate from the other demos.

Figure 6

As shown in **Figure 6**, if all goes well and we are happy with the results, it's time for drinks on Friday afternoon. If not, we do drinks anyway. Sometimes we also do a knowledge transfer session where someone shows what they have been working on lately.

After the weekend, we start our deployment week. Monday afternoon is the code cutoff for the deployment to the UAT environment. The team has until 3 p.m. to fix any bugs and retest. I recheck the release notes at this time to see whether all the issues listed have, in fact, been covered by this sprint and will make the deployment.

The deployment instructions involve a list of artifacts that need to be deployed as well as any additional instructions, forms, or workflow files that need to be uploaded to the portal manually. The instructions also include any manual configuration steps.

On Tuesday morning, our tech partner deploys our code to the UAT environment. When UAT is signed off by the product owner, the code is deployed to the Clones environment (which is a second acceptance environment that contains an exact copy of live data from the production server) on Wednesday. On the Clones environment we test to verify that the live data will not be affected by our changes and to see whether the added functionality is doing what it should do with the actual data.

van der Pal: So what happens if you are testing on the Clones environment and something doesn't work as expected?

Vermeer-Ooms: When our testing finds a major bug, the bug should be fixed as soon as possible. We then exchange e-mail with the tech partner to agree on a new deployment date as soon as possible.

After the new deployment is done, we retest and repeat the process if more fixes are needed. Sprints such as this necessitate a lot of flexibility from our tech part-

ner, and we are lucky that they are usually very helpful in getting done what we need.

ten Buggencate: Now that everything is done, do you also have a sprint review to see what you can do better next time?

Vermeer-Ooms: Because the deployment to the live environment is on Thursday evening, I often plan to have the sprint retrospective meeting on this day. In most cases, there is not much left to do for the sprint, which means it is a good time for us to reflect on what went well during the sprint and what can be improved for future sprints.

Our retrospective meetings are open to others in the company, but it is important that my team members feel safe enough to speak their minds, tell each other their concerns and, of course, give each other compliments (see **Figure 7**).

I ask each member of the team to prepare for this meeting by writing down the good stuff and the bad stuff they encountered during this sprint. Before the meeting, I prepare a shared Google doc where they can write these points under their name. I also list the actions that came out of the previous retrospective meeting, so they can see which actions have and have not been accomplished. We start the meeting with a round of bad stuff. Everybody is free to speak their mind as I take additional notes in the shared doc. When everybody has had their turn, we go through the items again and write down an action list in order to improve our way of working.

Figure 7

When everybody feels that their points have been heard, we do the same thing to discuss the good stuff. I make sure all comments are written down in the shared doc, because these are always nice to read later. Sometimes we also do a team demo during this meeting so everyone keeps up to date with the latest additions to the functionality of our product. Sometimes team members present what they have already been doing for the next sprint, or we discuss some of the technical discoveries that were made during the sprint. At the end of the demos, we always applaud.

If final testing of the Clones environment in the afternoon gives us the expected results, we give the go-ahead for the deployment to the live environment that night. After 6 p.m. the holding pages of the live site go up to show end users that the platform is undergoing maintenance. Deployment is done, and then we get the signal that we can go in to verify that the deployment was successful. If everything went OK, we are happy to take down the holding pages, because the sprint made it successfully to production.

van der Pal: But this is only Thursday evening; what do you do with Friday?

Vermeer-Ooms: Next Monday we will be doing sprint planning for the next sprint, so I use Friday to get everything cleaned up and organized again. I go through all the JIRA issues that were planned for this sprint to see whether they have been reassigned and set to the correct status (resolved). There are always some issues that were not completely resolved during the sprint, so these need to be moved to the next sprint so they will be open for discussion during the upcoming sprint planning session. I put the actions that came from the retrospective meeting on a Post-it note on the Scrum board so it is clear that they also are tasks to be done.

> **SAY WHAT YOU THINK**
> Our retrospective meetings are open to others in the company, **but it is important that my team members feel safe enough to speak their minds,** tell each other their concerns and, of course, give each other compliments.

Sometimes I decide that the Scrum board needs some cleaning up. Any Post-it notes still hanging are taken down, and I redo the lines on the board. I like straight lines.

Friday is also a day that team members spend on their other projects, clean up their environments, take on the actions from the retrospective meeting, or take on a task that will improve the quality of our code or improve our process. If they have nothing to do (which has never happened), they can help me prepare for the next sprint planning.

ten Buggencate: Wow—so that's what a whole sprint looks like outside of a training room. Thank you for sharing! `</article>`

MORE ON TOPIC:

LEARN MORE

- The Scrum Guide

JOSH JUNEAU

BIO

HTML5 and JSF

Mix and match the two technologies to create sophisticated applications.

This article focuses on integrating HTML5 into new and existing Java EE applications using JavaServer Faces (JSF). It covers the basics of adding HTML5 components into existing JSF views and constructing entire JSF views using HTML5 markup.

This article also covers the development of server-side templates, making it easy to change the look and feel of different views within an application. You will see examples demonstrating techniques for sending data to and from the server via HTML5 components using WebSocket and JSON-P, and you will learn how to build these solutions using NetBeans IDE.

The examples in this article demonstrate how to generate the same web form using standard JSF markup, HTML5 only, and both JSF and HTML5. The examples also demonstrate how seamlessly HTML5 elements can be processed as JSF components, and vice versa. After reading this article, you should have the knowledge necessary to begin integrating HTML5 components and technologies into your JSF applications.

Development Tools and Techniques

HTML5-Compatible Markup

When developing a web application, there are many decisions that have to be made beforehand. What functionality will the application provide? What type of user will be working with the application? Which devices will be used to work with the application? These are just some of the questions that must be answered before you can choose which technology or web framework to use.

There are hundreds of frameworks and technologies that can be used for developing web applications today. JSF is a mature and robust specification that ranks among the leading choices. It provides a solid foundation for sophisticated server-side applications that have the ability to maintain state, and it allows developers to focus on building the application, rather than spending lots of time developing JavaScript and CSS. It does so by supplying a bevy of components that are ready to use for building sophisticated user interfaces, abstracting away the implementation details of each component.

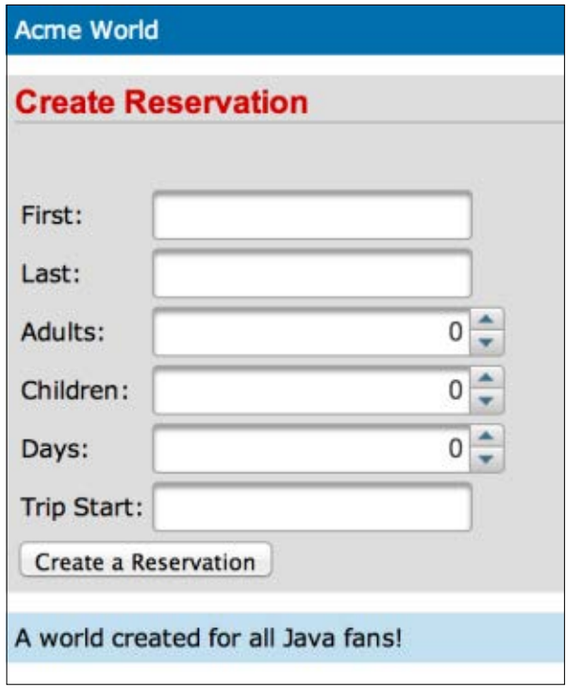


Figure 1

HTML5 is a new standard for developing portable applications that scale across devices seamlessly. JSF 2.2 enables interactivity between HTML5 and JSF views, allowing developers to take advantage of the scalability and modernization of HTML5, while maintaining solid grounds for a robust and dependable web application. JSF 2.2 lets developers choose to use JSF components, HTML5 elements, or both.

JSF 2.2 introduced new namespaces, which can be specified for supplying *pass-*

through elements, *pass-through attributes*, or both within HTML pages. Using these new features, HTML5 elements can be treated as JSF components, or JSF components can pass through to the browser without interpretation by JSF components or renderers. HTML5 pages can use JSF backing beans in the same manner as a standard JSF view, enabling seamless interactivity with JSF.

A Typical JSF View

Before we take a look at any HTML5, let's look at a standard JSF view. The examples used for this article are a subset of the application for the Java developer's paradise, Acme World Resort. In this article, we'll focus on a form that is used to book a reservation for a stay at the resort. The JSF-only view contains a mixture of standard JSF components and PrimeFaces components. It is also good to note that the view does not contain validation, but it is important to add validation to any web application prior to release.

YOUR CHOICE
JSF 2.2 lets developers choose to use JSF components, HTML5 elements, or both.

Listing 1 contains the JSF-only view for the reservation form. As you can see, the view contains a number of PrimeFaces components, specifically inputText, spinner, and calendar, to formulate the reservation form. **Figure 1** shows what the JSF-only view looks like.

HTML5-Only View

Now that we've seen how to produce the form using JSF markup only, let's shift to HTML5. The form in **Listings 2a**, **2b**, and **2c** contains a mixture of JSF Facelets and HTML5 markup. The Facelets markup is used for template purposes only, and if the page contained no Facelets markup, it would still function properly with the JSF engine. Pass-through elements were introduced with JSF 2.2, and they are the reason that the HTML5 elements in **Listings 2a**, **2b**, and **2c** are processed as JSF components.

To get started using HTML5 elements with JSF, add the jsf namespace for the pass-through elements (xmlns:jsf="http://xmlns.jcp.org/jsf") to the page. The jsf namespace can be added to any HTML5 element attribute, allowing that element to be processed by the JSF runtime. Take a look at the <head> element:

```
<head jsf:id="head">
```

The jsf:id="head" tells the JSF

LISTING 1 / LISTING 2a / LISTING 2b / LISTING 2c

Note: The following listing has been excerpted, as noted by the . . . symbol. The full code listing is available by downloading the code listings for this issue.

```
<h:form id=" parkReservationForm" >
  <h1>Create Reservation</h1>
  <br/>
  <h:messages id=" messages"
     infoStyle=" color: green;"
     errorStyle=" color: red;" />
 <br/>
 <h:panelGrid columns=" 2" >
  <label for=" firstName" >First:</label>
  <p:inputText id=" firstName"
      placeholder=" Enter First Name"
      value=" #{...firstName}" />
. . .

  <label for=" numAdults" >Adults:</label>
  <p:spinner id=" numAdults"  min=" 1"  max=" 15"
     value=" #{...numAdults}" />
. . .

  <label for=" startDate" >Trip Start:</label>
  <p:calendar id=" startDate"
     value=" #{...tripStartDate}" />

 </h:panelGrid>

 <h:commandButton id=" parkReservation"
 action=" #{parkReservationController.createReservation}"
 value=" Create a Reservation" >

 </h:commandButton>
</h:form>
```

Download all listings in this issue as text

engine to interpret this component as <h:head>. Similarly, reviewing an input component, we can see that attributes essential for JSF processing are passed through to the JSF runtime. In the following inputText element, the id and value are passed through:

```
<input type="text"
      jsf:id="firstName"
      jsf:value="#{...}"/>
```

As a result of the pass-through elements, the HTML5 element is treated as a first-class JSF h:inputText component that is associated with a server-side UIComponent instance. To be treated as a JSF component, at least one element must contain the jsf pass-through namespace. This allows standard HTML5 elements to accept Expression Language (EL) to retrieve and set managed bean properties. Therefore, when the form is submitted (or element values are sent to the server via JavaScript), the values are sent to the managed bean and processed accordingly.

Figure 2 shows what the HTML5 page might look like. Note that this code uses the native HTML5 calendar element, which might be rendered differently across various browsers.

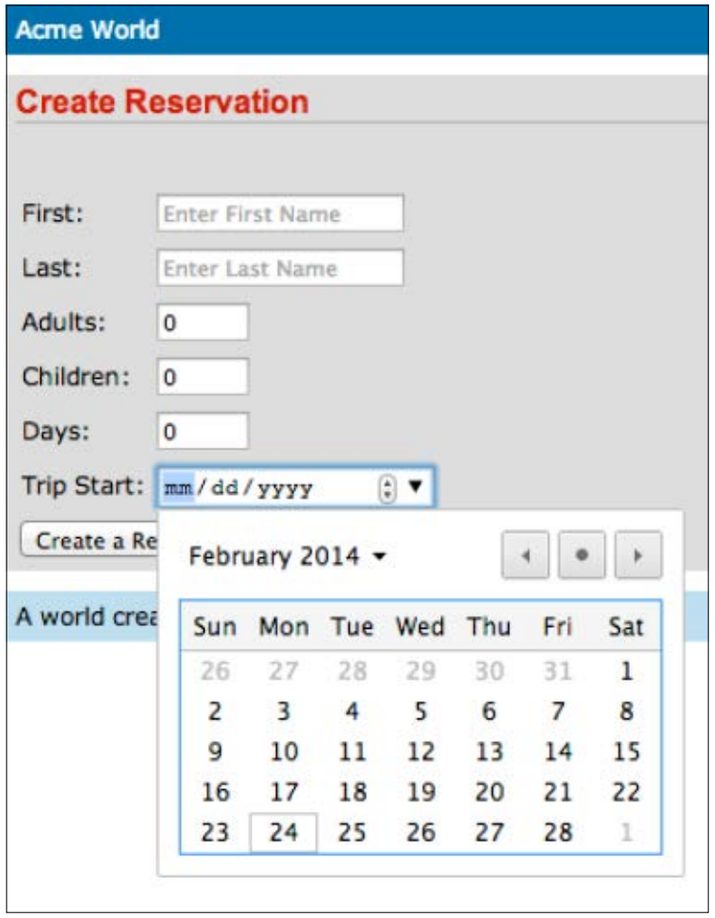


Figure 2

Sprinkling HTML5 into Existing JSF

JSF and HTML5 play nicely together. In fact, an HTML5 element can be added into any existing JSF view and function just as if it were a standard JSF component. However, in some cases, we might wish to use JSF components,

LISTING 3

Note: The following listing has been excerpted, as noted by the . . . symbol. The full code listing is available by downloading the code listings for this issue.

```
<h:form id=" parkReservationForm"  prependId=" false" >
 <h1>Create Reservation</h1>
 <br/>
 <h:messages id=" messages"  infoStyle=" color: green;"
    errorStyle=" color: red;" />
 <br/>
 <h:panelGrid columns=" 2" >
  <label for=" firstName" >First:</label>
  <h:inputText id=" firstName"
      p:placeholder=" Enter First Name"
      value=" #{...firstName}" />
. . .

  <label for=" numAdults" >Adults:</label>
  <h:inputText id=" numAdults"  p:type=" number"
      p:min=" 1"  p:max=" 15"
      value=" #{...numAdults}" />
. . .

  <label for=" startDate" >Trip Start:</label>
  <h:inputText p:type=" date"  id=" startDate"
      value=" #{...tripStartDate}" >
   <f:convertDateTime pattern=" YYYY-MM-dd" />
  </h:inputText>

 </h:panelGrid>

 <h:commandButton id=" parkReservation"
      action=" #{...createReservation}"
      value=" Create a Reservation" >

 </h:commandButton>
</h:form>
```

Download all listings in this issue as text

but *enhance* them by specifying attributes that are available for use only via the component's HTML5 counterpart. Such a solution is made possible using JSF 2.2's addition of the *pass-through attributes*. Pass-through attributes are the converse of pass-through elements: a pass-through attribute is applied to a JSF component to signify that the specified attribute should be ignored by the JSF runtime and passed directly through to the browser. The resulting element will be the HTML5 equivalent of the JSF component, including all attributes. This capability allows you to specify any number of pass-through attributes for a JSF component, and they will be completely ignored by JSF but rendered as HTML5 in the browser. This is a very handy feature, because HTML5 is still evolving and new attributes could be added in the future. The pass-through attribute feature of JSF 2.2 enables the specification of any current or future HTML5 attribute on a standard JSF component.

There are a few different ways to use pass-through attributes within your JSF views. One technique is to declare the pass-through namespace (xmlns:p="http://xmlns.jcp.org/jsf/passthrough") within the JSF view, and then precede any HTML5 attributes within a JSF component with the namespace prefix.

In **Listing 3**, the same form that was generated in the previous sections has been reimplemented using JSF components with pass-through attributes. Reviewing the first inputText component in the view, the HTML5 placeholder attribute has been specified, which allows a textual hint to be added to the text box on the rendered view. Because the placeholder attribute is not available on the input Text component, it must be passed through to the HTML5 rendering engine:

```
<h:inputText ...
 p:placeholder="text here"...
```

Similarly, pass-through attributes are used with inputText components to signify the type of component that should be rendered. In this case, HTML5 number and date elements will be rendered:

```
<h:inputText ...
 p:type="number"...
...
<h:inputText ...
 p:type="date"...
```

If only a single pass-through attribute is needed, it is possible to specify it by nesting the f:passThroughAttribute tag within a component tag. **Listing 4** demonstrates how to use the f:passThroughAttribute tag to specify the type of an inputText component to render an HTML5 date element. The f:passThroughAttribute tag accepts name and value attributes to specify the type of attribute along with the value.

To specify more than one attribute using this same technique, nest the f:passThroughAttributes tag within a component tag, and use an EL expression to specify the value as a Map<String, Object> within the managed bean. The Map should be composed of a name/value pair for each attribute to be used. **Listing 5** demonstrates how to construct such a Map, and **Listing 6** demonstrates how to specify the f:passThroughAttributes tag to render an HTML5 number element. All of the resulting forms will look similar to **Figure 2**.

LISTING 4 LISTING 5 / LISTING 6

```
<h:inputText id=" startDate"  value=" #{parkReservationCon-
troller...tripStartDate}" >
 <f:passThroughAttribute name=" type"  value=" date" />
</h:inputText>
```

Download all listings in this issue as text

Stateless Views

By default, JSF stores the state of all UI components after each request. With HTML5 and some of the popular web frameworks today, state is sometimes becoming less of a requirement. For instance, if you are creating a login page, you typically do not wish to persist stateful values in memory. Rather, you simply want to pass the values through for processing, and then move onto the next task. A new feature in JSF 2.2 is the ability to create stateless views. You can choose to mark a view as stateless by including the transient attribute on the f:view tag, and specifying true as its value:

```
<f:view transient="true">
```

If you are using a stateless view, it is important to use the proper session management within the managed bean. Therefore,

@RequestScoped should be specified on a managed bean, rather than @ViewScoped or @SessionScoped.

Note: Be sure to use the javax.enterprise.context.RequestScoped annotation for a CDI application.

Server-Side Templating

A significant feature of JSF is its ability to perform server-side templating. JSF ships with Facelets technology, which provides support for developing templates that can be applied across different views within an application. It can be advantageous to use server-side templating, even if you are developing an application that harnesses the client, such as an HTML5/JavaScript front end. The JSF 2.2 feature known as Resource Library Contracts provides the ability to supply an alternative look and feel for different portions of one or more applications, without the need to use multiple templates for each.

To make use of Resource Library Contracts, specify a contracts folder within your JSF application. Create one or more named contracts (directories) within that folder. Each contract should contain resources that are required for supplying the look and feel for that contract. For instance, suppose that the Acme World website had a different look and feel for those logged in as administrators. You could create two different contracts, by following an application structure such as that shown in **Figure 3**.

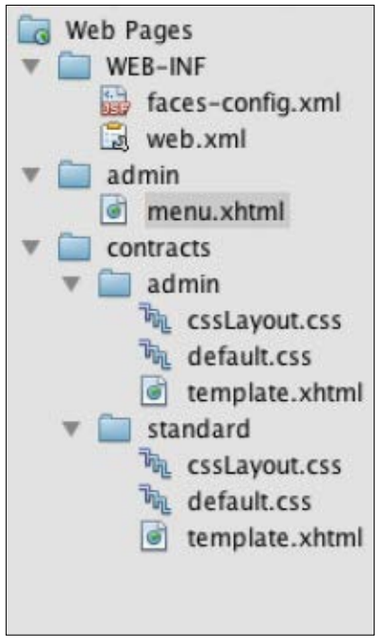


Figure 3

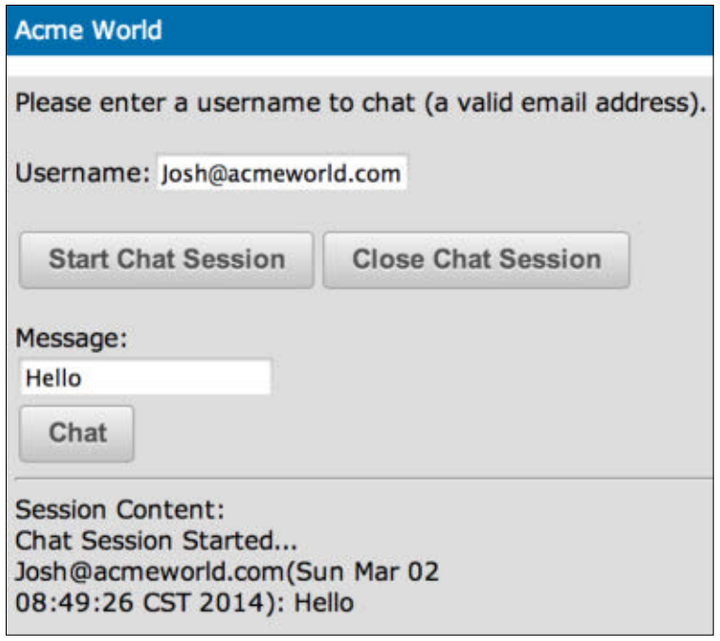


Figure 4

LISTING 7

```
<resource-library-contracts>
 <contract-mapping>
  <url-pattern>/admin/*</url-pattern>
  <contracts>admin</contracts>
 </contract-mapping>
 <contract-mapping>
  <url-pattern>*</url-pattern>
  <contracts>standard</contracts>
 </contract-mapping>
</resource-library-contracts>
```

Download all listings in this issue as text

To configure the contracts, specify the resource-library-contracts element within the faces-config.xml file accordingly. **Listing 7** demonstrates how to specify the configuration to use different templates for both the standard and admin contracts in our example. All pages can specify the same template name, and the resource library contract handles the job of determining which template to apply:

```
<ui:composition
template="/template.xhtml">
```

The template can also be packaged up in a JAR file so that it can be applied to more than one application, if desired. For more information, see the Resource Library Contracts information in the Java EE 7 tutorial.

WebSocket, HTML5, and JSF

HTML5 solutions require fast communication, and WebSockets provide that communication channel. WebSockets allow for full-duplex communication over a single TCP connection. What does this mean? Instead of sending separate requests for each transmission, a connection can be opened for two-way communication, remain open while needed, and then be closed when the communication is complete.

This section is not a primer on WebSocket; to learn details regarding the technology, please refer to the WebSocket section of the Java EE 7 tutorial. Instead, this section will demonstrate how to develop an application that uses a combination of JSF and HTML5 for the chat web view, along with WebSocket for communication.

In this example, we'll take a look at a simple chat view that uses a WebSocket connection to openly broadcast messages to all sessions connected to the same server endpoint. The simplistic chat application is shown in **Figure 4**.

Every WebSocket transmission follows a similar procedure: open the WebSocket connection, send messages from various clients to the WebSocket endpoint, and then close the connection. In our example, the user types in a chat username, and then clicks the Start Chat Session button to open a WebSocket communication channel. Once that has been completed, the user can type messages and send them to the chat room. After the connection is established, the user will begin to see any messages sent to the WebSocket endpoint from other users.

Listing 8 shows the source code for the simple chat room Facelets view, which contains three PrimeFaces commandButton components and a couple of HTML5 text input elements. This mixture of JSF and HTML5 is used for interacting with the JSF backing bean and WebSocket endpoint, and the output of the chat (returned from the endpoint) is displayed within a div near the bottom of the form.

The PrimeFaces command Button components communicate with the JSF runtime via action attributes, which are bound to a backing bean identified as ChatController. They also communicate with the WebSocket connection via JavaScript using the onclick event. **Listings 9a** and **9b** show the JavaScript that is invoked via the button events. For instance, when the Chat button is clicked, the sendChatMessage() JavaScript function is invoked, sending the message contained within the corresponding text box to the WebSocket endpoint.

The WebSocket endpoint class is identified as ChatServerEndpoint (see **Listings 10a** and **10b**). The @ServerEndpoint annotation marks the class as a WebSocket endpoint, and the encoders and decoders attributes are used to list the classes that can be used to translate the message for use. The value attribute contains a path that specifies the endpoint that is made accessible to clients. In this case, the WebSocket is accessible at ws://localhost:8080/JavaMagazine-HTML5JSF/chat.

The sendChatMessage() client JavaScript function encodes the message into a JSON object by stringing together the username and message into a JSON string of name/value pairs. It then sends the JSON string to the ChatServerEndpoint. When the

LISTING 8 / LISTING 9a / LISTING 9b / LISTING 10a / LISTING 10b

```
<form jsf:id=" chatForm"  jsf:prependId=" false" >
 Please enter a username to chat (a valid email address).
 <br/><br/>
 <label>Username:</label>
 <input type=" text"  jsf:id=" username"
   jsf:value=" #{chatController.current.username}" />
 <br/><br/>
 <p:commandButton onclick=" initiateChatSession();"
     update=" sendMessage jsfOutput"
     value=" Start Chat Session"
     action=" #{chatController.startSession}" />
 <p:commandButton onclick=" closeChatSession()"
     update=" sendMessage jsfOutput"
     value=" Close Chat Session"
     action=" #{chatController.closeSession}" />
 <br/><br/>
 Message:<br/>
 <input type=" text"  jsf:id=" message"
   jsf:value=" #{chatController.current.message}" />
 <br/>
 <p:commandButton value=" Chat"  id=" sendMessage"
     disabled=" #{!chatController.sessionOpen}"
     onclick=" sendChatMessage()"
     action=" #{chatController.sendMessage}" />
 <hr/>
 Session Content:<br/>
 <div id=" output"  class=" chatOutput" >
  <h:outputText id=" jsfOutput"
     value=" #{chatController.chatOutput}" />
 </div>

</form>
```

Download all listings in this issue as text

endpoint receives the message, the method annotated with @OnMessage is invoked, but it does not perform any message translation, because a decoder (see **Listing 11**) is used to parse the object via the JSON-P API upon message receipt, and an encoder (see **Listing 12**) is used to put the resulting message into a presentable String to transmit to all clients.

The ChatServerEndpoint class maintains a list of open WebSocket sessions, and each time a message is sent to the endpoint, it is broadcast to each session. The ChatController CDI backing bean (see **Listing 13**) is used for loading the current chat message into the session scope for further use, if needed. The username is also stored within the bean for any further use, and the bean maintains a Boolean indicating whether the chat session is active.

This simple chat client demonstrates that both HTML5 and JSF can be used while working with WebSocket. You are encouraged to visit the Java EE 7 tutorial to learn more about WebSocket so that you can begin to create sophisticated solutions using this technology.

HTML5 Development with NetBeans IDE

The NetBeans IDE makes it particularly easy to work with HTML5.

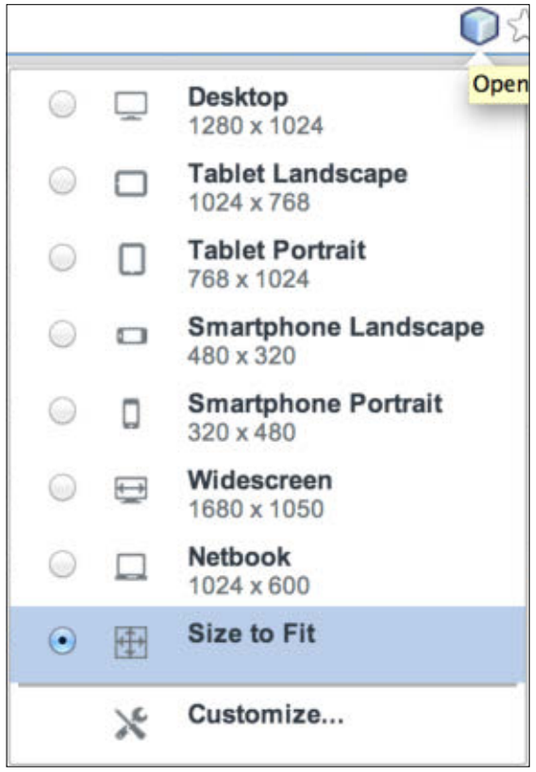


Figure 5

For starters, when the NetBeans Connector extension is installed into a Chrome browser, a NetBeans action menu is added to Chrome, enabling features for making HTML5 development more productive. NetBeans provides a live preview of web pages, which allows automatic redeployment of pages upon save. Users can save changes and see the changes applied immediately within the web page if they are using Chrome or they are on mobile devices running Android or iOS. This feature enables automatic page refresh while making modifications within HTML pages or JSF views alike.

LISTING 11 LISTING 12 / LISTING 13

```
public class ChatDecoder
 implements Decoder.Text<Chat> {
 @Override
 public void init(final EndpointConfig config) {
 }

 @Override
 public void destroy() {
 }

 @Override
 public Chat decode(final String textMessage)
   throws DecodeException {
  Chat chatMessage = new Chat();
  JsonObject obj = Json.createReader(
    new StringReader(textMessage))
    .readObject();
  chatMessage.setMessage(obj.getString("message" ));
  chatMessage.setUsername(obj.getString("username" ));
  chatMessage.setChatDate(new Date());
  System.out.println("decoder..." );
  return chatMessage;
 }

 @Override
 public boolean willDecode(final String s) {
  return true;
 }
}
```

Download all listings in this issue as text

Speaking of mobile development, NetBeans provides responsive web design capability, allowing developers to choose the layout of choice within the browser plugin (see **Figure 5**). Using the plugin, content can be dynamically reformatted to size, depending upon the selected dimensions. This enables developers to see what their applications

might look like across the screens of different devices.

NetBeans provides excellent code completion when working with HTML5 elements, JSF components (even if you are using a library such as PrimeFaces), JavaScript, CSS, and more. This makes development more productive; rather than parsing through documentation, the available options can be displayed onscreen as code is typed. For instance, when typing CSS tags, pressing the Ctrl and space keys simultaneously will provide CSS rule code completion (see **Figure 6**).

The NetBeans IDE also includes an HTML5 project type. These projects enable full support for libraries such as AngularJS, Knockout, and more. It is even simple to develop a Cordova application and deploy directly to a mobile device. These are just a few of the options that are made available to developers using NetBeans. NetBeans IDE is one of the best choices available for HTML5 and JSF development.

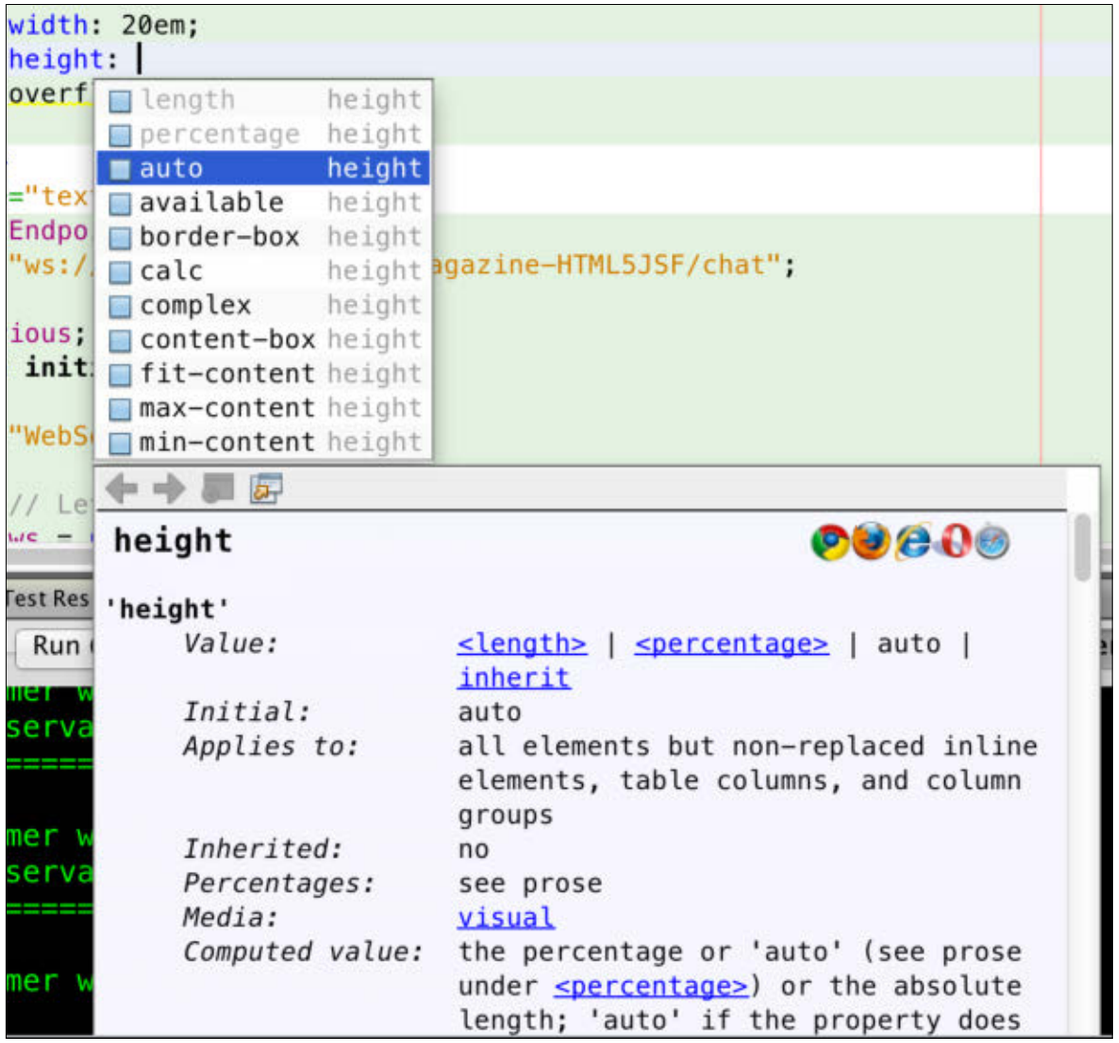


Figure 6

Conclusion

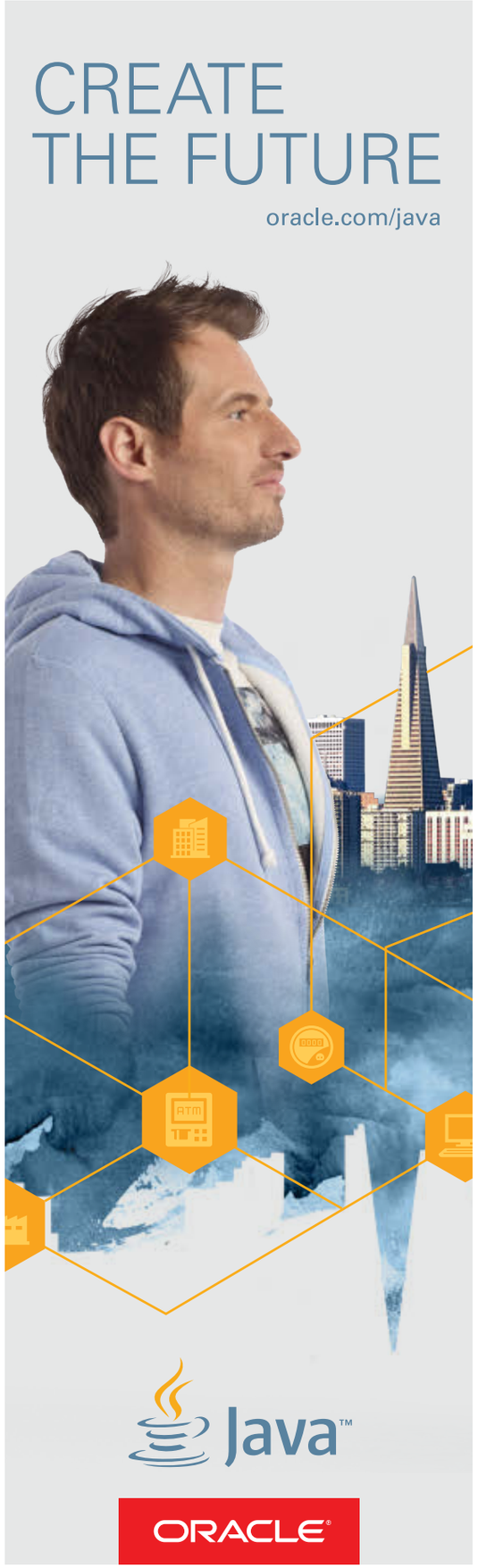
JSF plays an important role in Java web development. With the release of Java EE 7, JSF has been enhanced to provide seamless integration with HTML5, enabling JSF developers to take advantage of the features that HTML5 has to offer, and vice versa. Developers can mix and match the two technologies to suit their needs, enabling the development of sophisticated applications.

JSF also works well with HTML5-geared APIs such as WebSocket and JSON-P. This article demonstrated solutions integrating these technologies, and showed how to harness an IDE such as NetBeans for building these solutions. **</article>**

MORE ON TOPIC:

LEARN MORE

- HTML5 section of the Java EE 7 tutorial
- NetBeans HTML5 support
- PrimeFaces



HARSHAD OAK

BIO

Build with NetBeans IDE, Deploy to Oracle Java Cloud Service

Save time and effort deploying applications.

In a previous *Java Magazine* article, I described how you could go about choosing a Java platform-as-a-service (PaaS) provider and introduced Oracle Java Cloud Service. In this article, we will look at how you can swiftly build and deploy applications to Oracle Java Cloud Service using NetBeans IDE.

Most Java PaaS platforms provide web dashboards with the ability to deploy applications packaged into WAR and EAR files. PaaS providers also provide Ant and Maven tools to help streamline the build and deployment process. Some, such as Oracle Java Cloud Service, go a step further and provide integration with multiple popular IDEs such as NetBeans, Eclipse, and Oracle JDeveloper. IDE integration is great for developers because it can lead to significant savings of time and effort. In this article, we will focus on using the new NetBeans IDE 8 with Oracle Java Cloud Service.

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

Development Tools and Techniques

Installing the Oracle Java Cloud Service SDK

Before we get to the NetBeans integration, we first need to set up the Oracle Java Cloud Service Software Development Kit (SDK), because all IDEs use the SDK to interact with Oracle Java Cloud Service.

The SDK comes with tools to help you develop, deploy, and manage applications on Oracle Java Cloud Service. For example, it provides commands for interacting with Oracle Java Cloud Service, a whitelist tool for checking your application's deployment readiness, and Ant tasks and Maven plugins.

Note that the SDK is not a set of classes or libraries that you have to use in your application code.

To install the SDK, download the latest version from Oracle Technology Network and then extract the contents of the downloaded file to a directory.

> **TO THE CLOUD**
> It will only be a matter of time before we will all be building software using IDEs that **integrate with multiple cloud services.**

Getting a Trial Version of Oracle Java Cloud Service

Next, install a 30-day trial version of Oracle Java Cloud Service from cloud.oracle.com. Once your trial version is registered, you will receive information about the data center, the username, the password, and the identity domain for your trial. Keep that information, because you will need it when we explore how to use Oracle Java Cloud Service from NetBeans.

Note that the trial version also includes a trial of Oracle Database Cloud Service. So you do not need to get a separate trial version of Oracle Database Cloud Service.

Setting Up NetBeans

One of the many nice features of NetBeans is its easy installation. Download either the Java EE version or the

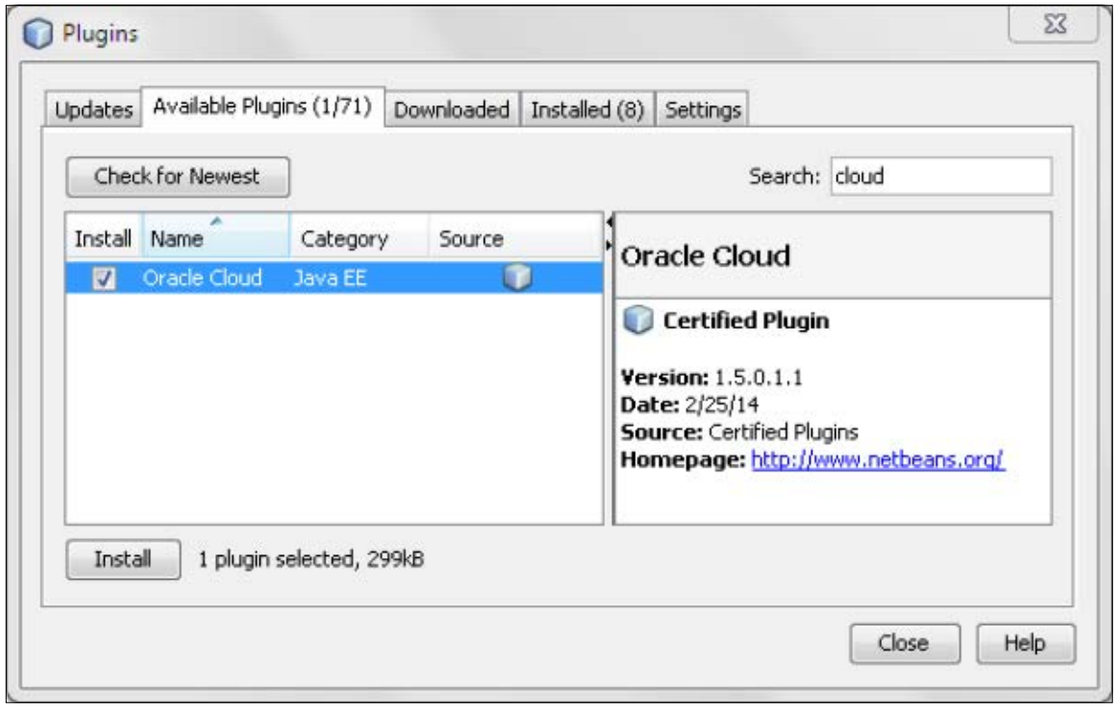


Figure 1

"All" version of NetBeans from netbeans.org. Install NetBeans and you are ready to go with all the standard capabilities.

Having said that, installing even the "All" version does not mean that all features will be activated right away and start taking up resources. NetBeans has a neat "Feature On Demand" capability that enables a technology only when you actually use it.

Installing the Oracle Cloud Plugin

Once NetBeans is set up, we next need to install the Oracle Cloud plugin for NetBeans. To do that, in NetBeans click **Tools -> Plugins** and then click the **Available Plugins** tab. There, search for cloud and you will see the plugin listed, as shown in **Figure 1**. Ensure that the checkbox for the plugin is selected and then click **Install**. Then follow the steps in the NetBeans IDE Plugin Installer to install the Oracle Cloud plugin.

Once the plugin is installed, you will find that a new "Cloud" section has been added to the Services window, as shown in **Figure 2**. If you are unable to see the Services window, enable it by clicking **Window -> Services** in the NetBeans menu.

Note that you can always use the **Window -> Reset Windows** option in the NetBeans menu to get back the default windows and the familiar look and feel.

Figure 2

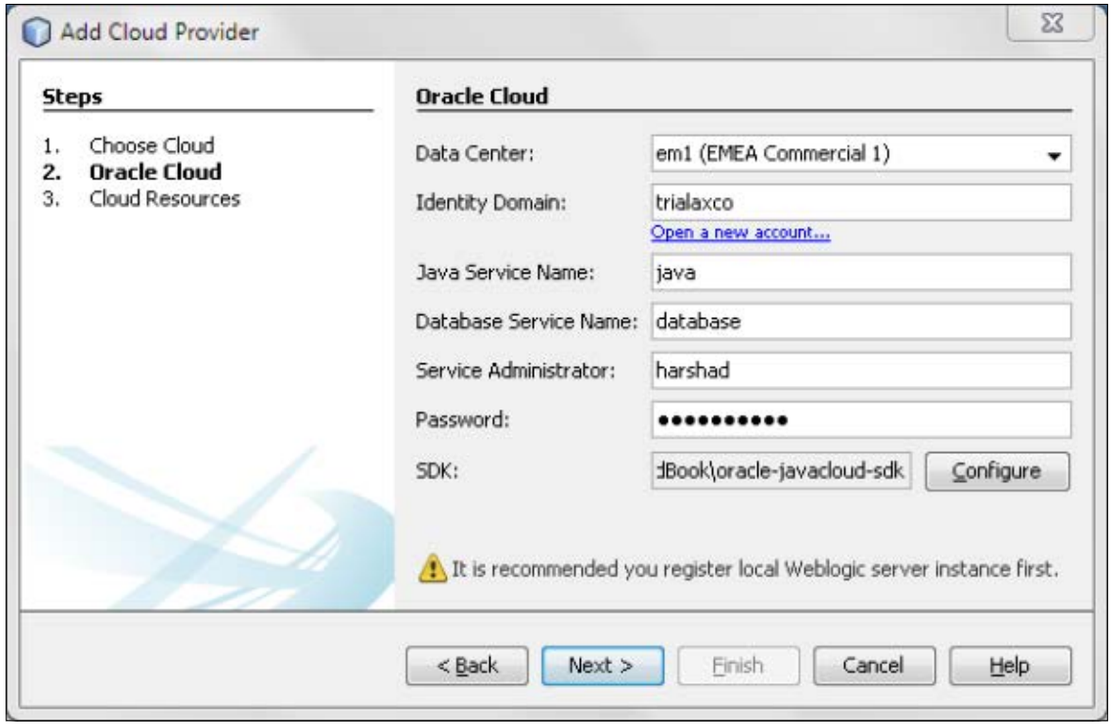


Figure 3

Now, right-click **Cloud** and select **Add Cloud**. You will see the Add Cloud Provider screen. Select **Oracle Cloud** and then enter the information you got when you installed the trial version of Oracle Java Cloud Service, as shown in **Figure 3**. Note that in the **SDK** field, you need to specify the path to where you extracted the Oracle Java Cloud Service SDK. Also, if your data center is not listed in the **Data Center** list, type in your data center's name.

As seen in **Figure 3**, NetBeans recommends that you register a local instance of Oracle WebLogic Server, which you can use during development. There's nothing stopping you from using the cloud environment even during development; however, it will be faster and easier to deploy and debug code locally than in the cloud. Oracle

Java Cloud Service runs Oracle WebLogic Server 11*g*. Download it here and then install it, so you will have a local server running.

Once your cloud is set up, you will find a new server and a "welcome app" listed, as shown in **Figure 4**. You can right-click the app to view, stop, or undeploy it. You can also access the Jobs Log and the Instance Log from within NetBeans by right-clicking **Oracle Cloud** and then clicking **View Jobs and Logs**.

Now that NetBeans is integrated with Oracle Java Cloud Service, let's next look at building a Java EE application that will use Oracle Java Cloud Service as well as Oracle Database Cloud Service.

Note that if you enabled the "Feature On Demand" capability while installing NetBeans and this is your first use of the Java EE capabilities of NetBeans, it might take NetBeans some time to activate Java EE. Also, if you chose to install Apache Tomcat or GlassFish during installation, you will find them also now listed under **Servers** in the Services window.

Building a Java EE Application

First, select **New -> Java EE -> Enterprise Application**. Enter JavaMag for the name of the project and click **Next**. In the New Enterprise Application screen,

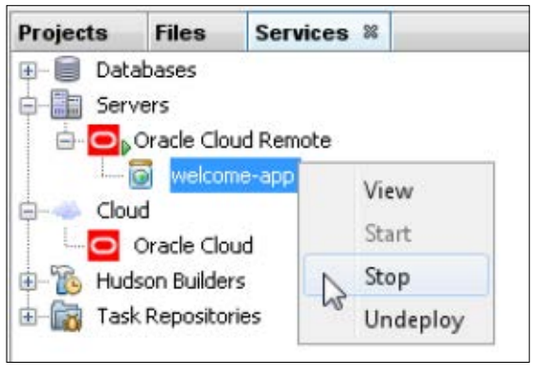


Figure 4

which is shown in **Figure 5**, select **Oracle Cloud Remote** from the **Server** list.

Oracle Java Cloud Service currently supports a mix of technologies from Java EE 5 and Java EE 6, as described here. NetBeans selects Java EE 5 and a source level of 1.5.

Then select the **Create EJB Module** and **Create Web Application Module** checkboxes. Click **Finish** and NetBeans will create the modules.

In our application, we will use a Java Persistence API (JPA) entity, a stateless session Enterprise JavaBean (EJB), and a servlet to create a new magazine record in the database.

We will begin by creating a new JPA entity named Magazine. Right-click the JavaMag-ejb project and select **New -> Persistence -> Entity Class**. As shown in **Figure 6**, enter Magazine in the **Class Name** field, select **entities** from the **Package** list, and ensure that the

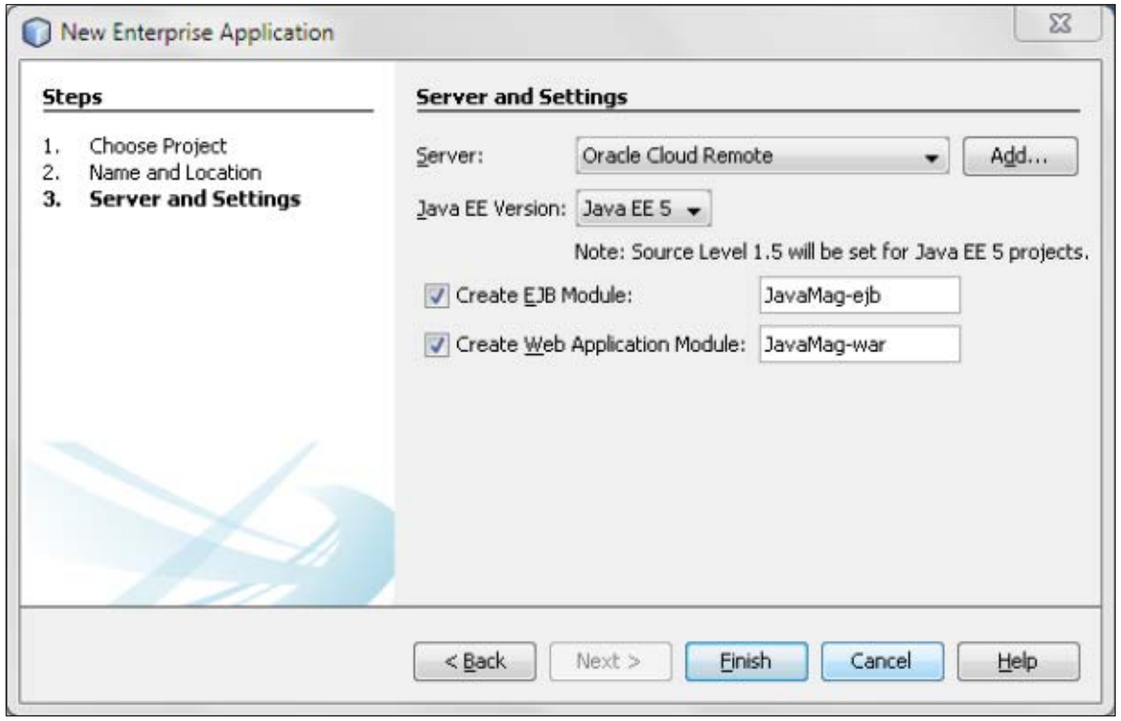


Figure 5

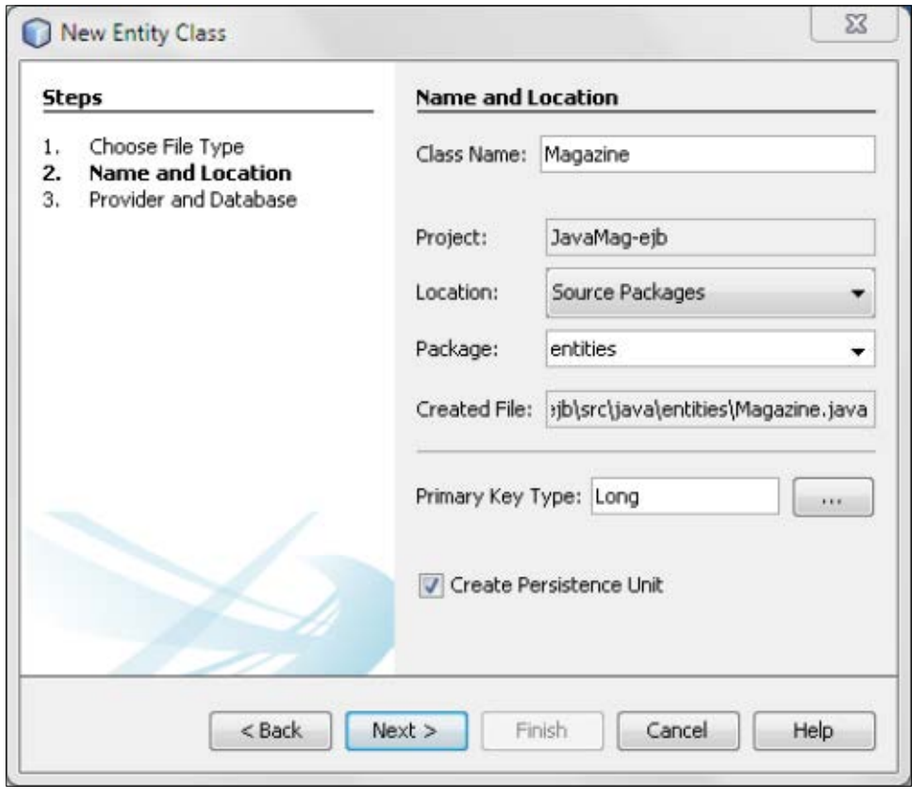


Figure 6

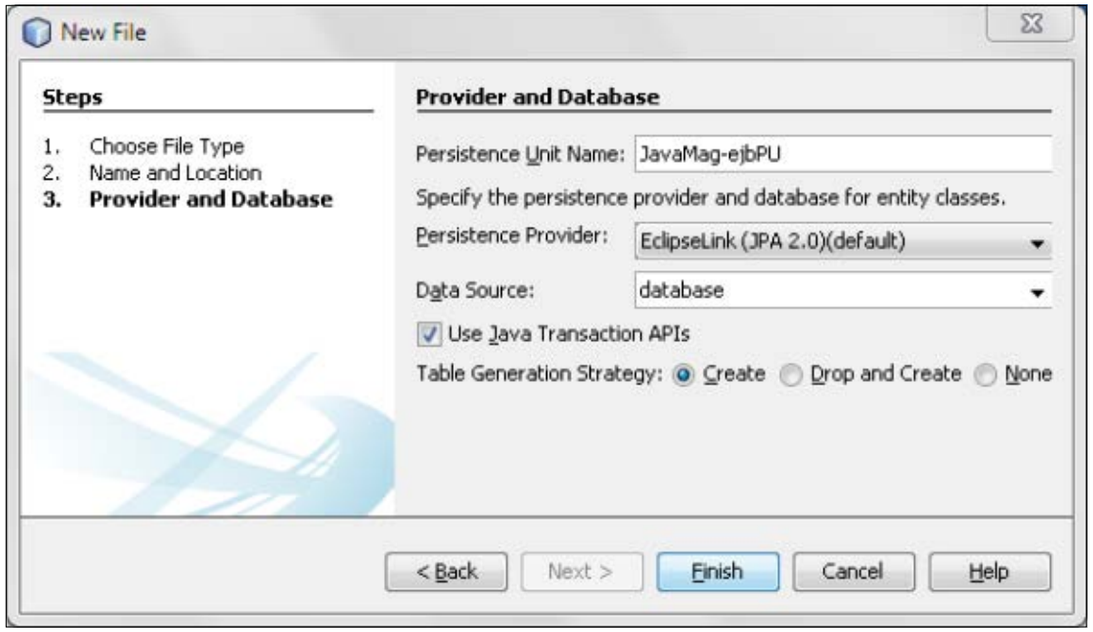


Figure 7

Create Persistence Unit checkbox is selected. Then click **Next**.

In the New File screen, which is shown in **Figure 7**, enter the **Data Source** name for the trial version of Oracle Database Cloud Service that you got with your trial version of Oracle Java Cloud Service. You will see this name listed in the **Data Sources** section upon logging in to the Oracle Java Cloud Service web-based dashboard.

Oracle Java Cloud Service supports JPA 2.0, so EclipseLink for JPA 2.0 is selected. Note that the **Create** option is the default selection for **Table Generation Strategy**, so a new Magazine table will be created when we attempt to persist new magazine information using the Magazine JPA entity.

NetBeans will generate the entity class file Magazine.java in the package entities. It will also generate a persistence.xml file where you will find the persistence unit configuration.

As shown in **Listing 1**, which is an excerpt from Magazine.java, NetBeans will add all the requisite annotations to specify that the class is an entity. In **Listing 1**, the GenerationType.AUTO annotation for the id property means that the persistence provider will pick an appropriate strategy based on the database. EclipseLink typically will create a new table for the key generation, because that's a strategy that will work across databases.

Now let's add a name property to the Magazine entity. To do that, right-click in the code and select **Insert Code -> Add Property**. In the screen that opens, enter the property name as name and select the option to generate getter and setter methods.

Next, we will create a new stateless session bean through which we will use the Magazine entity. Right-click the **JavaMag-ejb** project and select **Enterprise JavaBeans -> Session Bean**.

As shown in **Figure 8**, enter AddMagazineBean for the name of the bean, select **ejb** from the **Package** list, and select **Stateless** for the session type.

Note that because Oracle Java Cloud Service supports EJB 3.0, you don't get the "no interface" option that was introduced in EJB 3.1. Also, Oracle Java Cloud Service supports only local EJB invocations, so select **Local** for the type of interface to create.

Click **Finish**. NetBeans will generate the AddMagazineBean

TIME-SAVER

Oracle Java Cloud Service **goes a step further** and provides integration with multiple popular IDEs.

LISTING 1 LISTING 2

```
@Entity
public class Magazine implements Serializable {
   private static final long serialVersionUID = 1L;
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private Long id;

   private String name;
```

Download all listings in this issue as text

bean along with the AddMagazineBeanLocal interface. Now select **Insert Code -> Add Business Method** to add a new addMagazine method to the session bean. Note that NetBeans will add the method signature to the local interface.

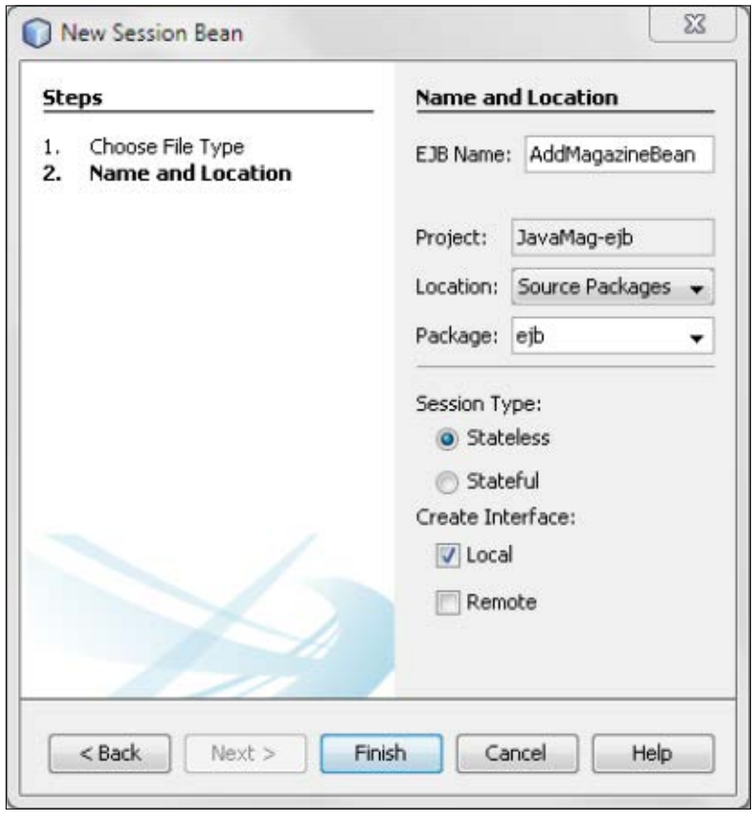


Figure 8

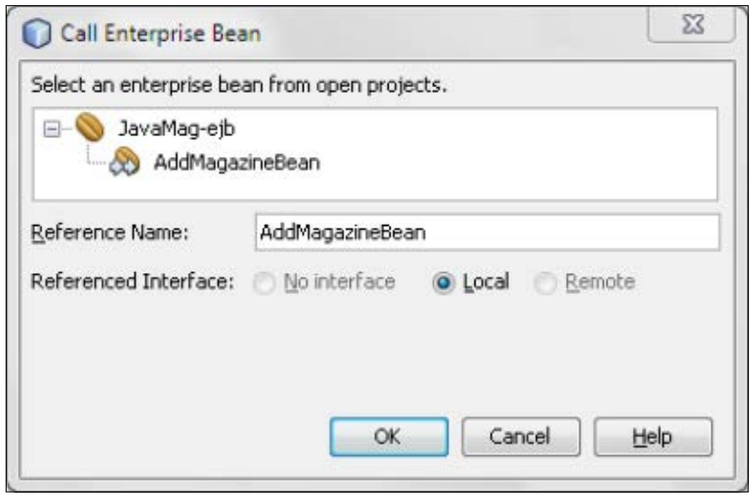


Figure 9

To add the new magazine to the database, we need to use the entity that we created earlier. To use the entity, we need to inject the EntityManager. To do so, right-click in the code and select **Insert Code -> Use Entity Manager**. NetBeans will add the dependency injection code for EntityManager as well as a persist method. We will use the persist method, as shown in **Listing 2**, and pass it an object of the Magazine entity.

We now have our entity and the session bean created in the JavaMag-ejb project. Next, we need to add a web component in the JavaMag-web project, which will use the AddMagazineBean to add a new magazine to the database.

Add a servlet to the JavaMag-web project by right-clicking the project and selecting **Web -> Servlet**. Name the servlet MagazineServlet and name the package servlets. Click **Finish**.

Now, to call the session bean from the servlet, right-click in the code and select **Insert Code -> Call Enterprise Bean**. Specify AddMagazineBean, as shown in **Figure 9**.

As shown in **Listing 3**, NetBeans will add the dependency injection with the @EJB annotation to the MagazineServlet.

We will next add a line in the processRequest method to call the bean and add a new magazine, as shown in **Listing 4**.

We now have our entity, the session bean, and the servlet in place, so we can deploy the application. Right-click the JavaMag enterprise application and click **Run**. Check the contents of the "JavaMag (run)" log in NetBeans and you will find that NetBeans built the WAR file for the web application module, built the JAR file for the EJB module, and packaged the two into an EAR file. It then uploaded the EAR file to Oracle Java Cloud Service.

As shown in **Figure 10**, the "Oracle Cloud Remote Deployment" log lists the activities being performed in the cloud. It shows that the cloud service first performs a virus scan, then a whitelist check, and then the actual deployment.

Once the deployment is complete, your default browser will open up and point to the index.jsp page in the application we deployed. Because we want to run the MagazineServlet, change the URL in the browser to /JavaMag-war/MagazineServlet to call the servlet. You will get output similar to what's shown in **Figure 11**.

By default, Oracle Java Cloud Service will want you to log in to your cloud service before you can access your servlet. However, if you wish to make the application public and you do not want any authentication, add an empty <login-config/> tag to the web.xml file.

Upon executing the servlet, select **Sql Workshop -> Object Browser** in Oracle Database Cloud Service. You will find that a new table named Magazine has been created and a

LISTING 3 LISTING 4

```
@EJB
private AddMagazineBeanLocal addMagazineBean;
```

Download all listings in this issue as text

```
JavaMag (run)    Oracle Cloud Remote Deployment

Uploading...
Deploying........
==================== Log file: virus-scan==========================

2014-03-11 00:39:12 CDT: Starting action "Virus Scan"
2014-03-11 00:39:12 CDT: Virus Scan started
2014-03-11 00:39:51 CDT: -------------------------------------------------------
2014-03-11 00:39:51 CDT: File Scanned:  "JavaMag.ear".
2014-03-11 00:39:51 CDT: File Size:     "9904765".
2014-03-11 00:39:51 CDT: File Status:   "CLEAN".
2014-03-11 00:39:51 CDT: -------------------------------------------------------
2014-03-11 00:39:51 CDT: Virus scan passed.
2014-03-11 00:39:51 CDT: "Virus Scan" complete: status SUCCESS

..
==================== Log file: whitelist===========================

2014-03-11 00:39:51 CDT: Starting action "API Whitelist"
```

Figure 10

```
Firefox    Servlet MagazineServlet    +
https://java-trialaxco.java.em1.oraclecloudapps.com/JavaMag-war/MagazineServlet

Servlet MagazineServlet at /JavaMag-war
New Magazine Added
```

Figure 11

new "Java Magazine" record has been inserted in the table.

Conclusion

In this article, we saw how to quickly set up and use NetBeans to develop a Java EE application and deploy it to Oracle Java Cloud Service. Considering the rapid adoption of cloud-based services, it will only be a matter of time before we will all be building software using IDEs that integrate with multiple cloud services for various stages in the software development process. `</article>`

MORE ON TOPIC:

LEARN MORE

- Oracle Java Cloud Service
- Oracle Database Cloud Service
- NetBeans IDE
- Oracle Java Cloud Service SDK

MICHAEL HÜTTERMANN

BIO

Mastering Binaries with Hudson, Maven, Git, Artifactory, and Bintray

A powerful tool chain that can be the backbone in a build/release and delivery pipeline

Development Tools and Techniques

In this article, we discuss patterns for mastering binaries in Maven-based projects. We differentiate between version control systems and component repositories as well as Maven releases and Maven snapshots. We talk about automatic releasing and explore a tool chain that integrates Hudson, Maven, Git, Artifactory, and Bintray to be a backbone in a build/release and delivery pipeline.

Now let's start by setting the stage with some important aspects of continuous integration (CI) and continuous delivery (CD).

CI and CD

CI includes code integrations that are run at least on a daily basis. The word *continuous*, as used in this context, denotes a repeatable process that occurs regularly and frequently. The word *integration* means that individually developed pieces of code that are stored in a source code repository are checked out as a whole; then they're compiled, packaged, tested, inspected, and deployed with build results integrated into web pages, sent out as an e-mail, or both.

Building and integrating software as soon as developers check in their changes is called *continuous build*. A *release build* (or a *release candidate build*) is often the build that pulls a specific version (which was tested before successfully) from the version control system (VCS) and creates a new baseline from that. The build server acts as a "single point of truth," so builds can be used with confidence for testing or as a production candidate.

With CD, you implement delivery pipelines, also called *build staging*, according to your corporate standards. For example, validating special quality requirements on higher build stages prevents code from being promoted before it's ready. The initial staging area is the developers' workspaces. Operationally, build pipelines often consist of different build jobs that are triggered automatically or manually and might have dependencies on each other. We talk about builds, but what exactly is a build?

Builds

A build is a standard, repeatable, and measurable compilation and packaging process done automatically. The specific steps of this process can vary. Some argue that building means compiling. Others might include in the process a preparation of the system, working on baselines in the VCS, compiling sources, running different sorts of tests, applying the "infrastructure as code" paradigm (for example, with Chef or Puppet), and packaging and distributing the configuration items. *Build* can also refer to the greater automation flow, including static code analysis, unit testing, and deployment. The output (the deliverables) of this process is often called a build, but from now on, we

> **TAKE NOTE**
> Be aware that **even early in the process,** you should use build environments that match those you'll use in production.

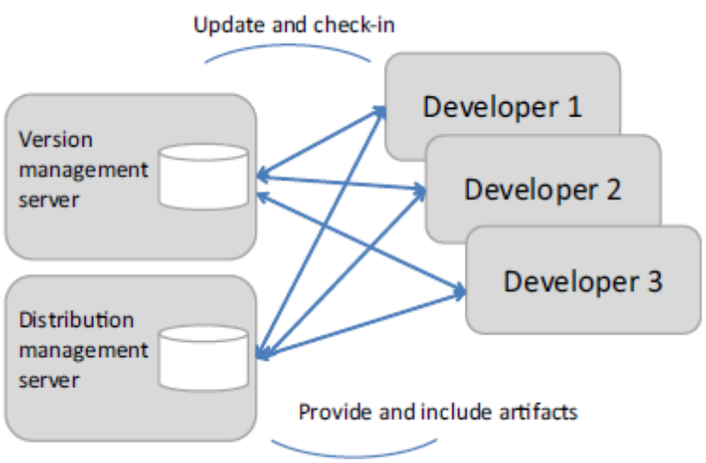


Figure 1

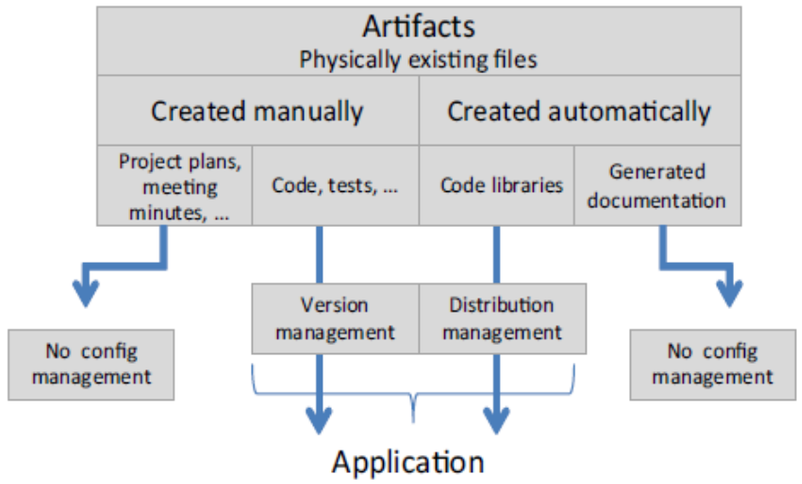


Figure 2

will call it an *artifact*.

The changes from multiple developers are integrated continuously and automatically. Each version of the build is the basis of later releases and distributions. As shown in **Figure 1**, a build—local and central ones—checks out and may update sources from the VCS and consumes and produces artifacts from and to a *component repository* (also called a *binary manager* or *distribution management*). Version control and distribution management are in place and are complementary.

The artifact is configurable and runs on different target environments, including Windows and Linux machines ("build once, configure anywhere"). Be aware that even early in the process, you should use environments that match those you'll use in production. With CD, promoting versions to higher staging environments is often a one-click event, where a lead engineer, a release manager, a deployment manager, or the domain expert pulls a new version to be deployed and configured to a defined target environment. An *environment* is all of the resources that your application needs to work and their configuration, as well as the hardware configuration (including CPUs, memory, and spindles) and the configuration of the operating system and middleware (including messaging systems and web servers). The term *infrastructure* summarizes all of the environments in your organization together with supporting services, such as firewalls and monitoring systems.

As shown in **Figure 2**, parts of each artifact are held in version control, and other parts are held in distribution management. The configuration management discipline calls all those artifacts *configuration items*. Configuration management decides whether to put artifacts into version control or into distribution management.

Now let's explore version control and distribution management (component repositories) in detail.

Version Control

Generally, it's important to archive your artifacts, but which types of artifacts you store where depends on the project context and the requirements. Coding artifacts—particularly source code, build scripts, and infrastructure as code—should be stored in the VCS. Although this sounds obvious, it's not always the case. Many projects, for example, patch their applications in production without having those changes under source control. In addition to source code, tests and build scripts need to be versioned as well as Puppet manifests or Chef cookbooks. The latter is often referred to as *DevOps*, although DevOps is much more than that.

I recommend that you externalize (and version) your runtime configuration settings. It's best practice to control all variable configuration values externally. Additionally, you should set up one central repository to store your assets, to avoid having multiple places where documentation might be located (for example, the VCS, a file sharing server, and your favorite e-mail system, all in parallel).

When you check your artifacts into the VCS, you'll have to decide how to arrange the files in the system and decide who works on which stream and why. The rule of thumb here is that you shouldn't open any additional streams for longer than necessary, even with a distributed VCS such as Git.

Component Repositories

Often it's necessary to reproduce software that's running on different machines. It can, therefore, be useful for a central release or configuration management department to label final versions of the software and put the build artifacts

into a specific build archive, often called a *definitive media library*. This ensures binary integrity, which means that the same deployment units are delivered to each target environment (there's no recompilation for further environments), and source code is always linked to produced binaries.

A component repository can also protect company assets and boost reusability. In the Java ecosystem, the binary versions are those from the standardized set of deployment units, such as JAR, WAR, and EAR files. Full-fledged component repositories offer many features including search capability, role-based permission systems, transactional handling of binary access, and much more. Now that we've differentiated between the VCS and component repositories, you're fit for automatic releasing.

Automatic Releasing

During automatic releasing, which is fundamental for a CD initiate, major parts of the release process are performed by scripts and tool chains. In this process, the whole team profits from automation and, under optimal conditions, a specific button is simply pressed to promote automatically created snapshot versions to release candidates or to promote release candidates to release status.

Prerequisites of a holistic automatic releasing process include

- Use highly integrated tool chains consisting of lightweight tools, for example, Hudson or Maven, that can be chosen and orchestrated as needed.
- Put configuration items (including sources, database scripts, middleware, infrastructure, configuration files—such as Java properties files—and build/deploy scripts) into version control.
- Wherever possible, use declarative formats (for example, Puppet manifests and Chef cookbooks) to set up the automatic releasing process.
- Declare (explicitly) and isolate dependencies of application, middleware, and infrastructure.
- Apply CI that continuously synchronizes the work of your colleagues.
- Distinguish between VCSs (such as Subversion and Git), where you hold your sources, and component repositories (such as Artifactory and Nexus), where you hold your software binaries.
- Build binaries once and deploy them to target systems by configuration (in other words, runtime configuration data is not packaged into the binaries in a static fashion; rather, the application is configured during deployment time or upon startup).
- Keep environments similar between development and operations (keeping them equal is not practical because of costs, benefits, and different nonfunctional requirements in specific environments).
- Define a process (and also one for patching and merging back changes in production versions).
- To ensure reproducibility, ensure that delivered software is solely built by the build server and is neither built manually nor patched or built on developers' desktops.

It's critical to understand that automation is important in order to gain fast feedback. Don't just automate because automating defined activities is so much fun, although this is a good motivator as well.

When running an automatic releasing process with Maven, you need to take special care of Maven snapshots and Maven releases, which we discuss next.

BEST PRACTICES

I recommend that you **externalize (and version) your runtime configuration settings.** It's best practice to control all variable configuration values externally.

Snapshots and Releases

First, it's important to consider build hygiene while releasing and working with Maven projects. For instance, a release build (in Maven terms, that is every version without a -SNAPSHOT in its version element) includes only frozen versions of both the produced and consumed artifacts. You might want to check this by using the convenient Maven Enforcer plugin, or you might want to adjust your project object model (POM) accordingly with the help of Maven's Versions plugin.

Let's make this even clearer by introducing a very small example. Let's imagine we want to play around with lambdas in a freshly built Java SE 8 application that looks like **Listing 1**.

To build this in a reproducible way, we just set up a minimalistic Maven POM, as shown in **Listing 2**. Take note of the compiler settings. We use Java 8 for both source and target to be able to use and compile the lambda expression.

Now look at the version element. We've introduced a variable, ${myVersion}, for the version that can be passed through while starting the Maven build. The parameterized version number is often a good approach during releas-

Figure 3

ing. We can just call a build—for example, with mvn clean install -DmyVersion=1.0.0-GA—and the produced binary will have the correct version. This way we can easily create releases candidates or release continuously, without the overhead of frequently patching the POM files with respective version numbers. The drawback is that the POM, as part of a baseline in your VCS, does not contain the concrete version number. This can be a showstopper, depending on project constraints and governance requirements.

Often a project team feels comfortable using Maven snapshots during development and providing releases for deployment to target environments. Take note: the term *deployment* is heavily overloaded. Please differentiate between deploying artifacts to a component repository—that means publishing binaries to it—and deploying artifacts to an application server (that is, the runtime container for the application). In this article, I don't cover any deployments of WAR files or EAR files to application servers. It's important to understand that although it's often fine to work with Maven snapshots during development and deploy them to a component repository, it's seldom a good idea to deploy snapshots to an application server on a higher integration test environment.

Let's now look at a different approach for what releasing can look like by designing an appropriate delivery pipeline. Keep in mind that for that, we've changed the POM file and replaced the version variable with 0.0.1-SNAPSHOT.

A Delivery Pipeline

Let's go through the essentials of the solution and discuss some common recipes for a Hudson build pipeline that has three build stages triggered continuously on each Git commit (see **Figure 3**). Be aware that, in practice, you'll probably want to set up a holistic pipeline, which might consist of multiple subpipelines. All build jobs have the Git integration configured pointing to the central master repository.

Stage 1. The first stage is triggered automatically on each commit in Git. Code is compiled, tested, packaged, and installed locally on the build server. Choose your build server topology wisely, for example, by scaling out horizontally and running builds only on build slaves, not on the master.

After the Maven build step mvn

LISTING 1 LISTING 2

```
package com.huettermann;

import java.lang.System;

/**
 * Hello world!
 */
public class App
{
  public static void main(String[] args) {
    Runnable r = () -> System.out.println("Hello World!");
    Thread t = new Thread(r);
    t.start();

  }
}
```

Download all listings in this issue as text

clean install runs successfully, the produced binary is placed in the folder named "target" of the build job's workspace and installed into the local Maven repository, which, depending on your configuration, is either in the home directory of the user running Hudson or another place local to your Hudson builds. According to the Maven approach, the artifact is named cat-0.0.1-SNAPSHOT.jar, which is the concatenation of the artifactId (cat), the version (0.0.1-SNAPSHOT), and the deployment type (.jar).

After we've produced the artifact, we can place it into an exchange medium for later reuse, for example, for sharing among developers, or later use between development and operations. Often, this is a component repository such as Artifactory, for both snapshots and releases, or Bintray, for releases only.

In our case, as part of the pipeline, we want to locally reuse the freshly created artifact in downstreamed build steps and check its quality before we distribute it to any teams. Thus, we go for the approach of temporarily archiving the artifact on the file system. We could also move and exchange artifacts across build slaves, but let's keep it simple here and focus on the scope of this article.

So let's copy the file to a transfer folder by squeezing some shell commands into a Hudson "execute shell" script build step, as shown in **Listing 3**.

Although there are options for versioning scripts and tracking changes in Hudson, for better maintainability, in real project life, you often want to put your shell scripts into the VCS as well. Because we later need the project version and the Git checkout hash, we now extract and store that information. Let's start with the version of the Maven-based project (see **Listing 4**), which is also part of the dedicated Hudson build step.

Many possible ways exist for extracting the version number. We've decided to use some sed commands, and store the version number in a Java properties file named version.properties. Given the version 0.0.1-SNAPSHOT, the file afterward includes the key/value pair version=0.0.1-SNAPSHOT.

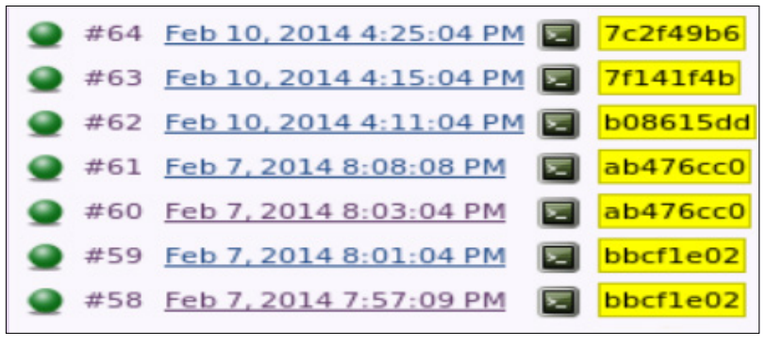


Figure 4

LISTING 3 LISTING 4 / LISTING 5

```
#!/bin/sh
rm -rf /home/michael/talk/transfer
mkdir /home/michael/talk/transfer
cp devops/target/*.jar /home/michael/talk/transfer
```

Download all listings in this issue as text

Because we later want to cherry-pick a tested version to promote to be a release, we also store and visualize the Git hash, that is, the commit that was built successfully by Hudson. For that, we've coded a Groovy postbuild action, similar to **Listing 5**.

This script adds the first eight characters of the hash to the build history of the build job for later reference; see **Figure 4**, which shows an enriched build history. Git commits at your fingertips.

Finally, Stage 1 triggers the downstream build job. In our case, this is Stage 2; thus, we add this build job to be Hudson's target destination of the "Trigger parameterized build on other project." Here, don't forget to pass through parameters you want to use later from inside downstream build jobs. In our case, we'll need the Git commit hash; thus, we configure to pass through the "Git commit that was built." Let's now move forward to Stage 2.

Stage 2. Stage 2 is an example dummy stage that illustrates any further activities on the previously built artifact or code baseline. Please imagine some heavy processing here, testing, and many more helpful things.

If you need to access the sources from version control, it's important to not rely only on Hudson's Git plugin. It's more stable to trigger a git checkout $GIT_COMMIT as the first build step in that particular build job of Stage 2. This ensures that the build job does work on exactly the same Git commit as the build job of Stage 1.

Closing Stage 2, we trigger Stage 3 with the same approach that we used to call Stage 2 from Stage 1. Additionally, we now add "Parameters from properties file" while configuring the "Trigger parameterized build on other project" section. Here, we use the properties from file /home/michael/talk/transfer/version.properties, which was created during Stage 1.

It's sometimes a bit tricky to make context information and environment variables available in Hudson build jobs. By passing the properties file, its key/value are made available to the downstream build job dynamically.

Stage 3. Our last stage contains the integration with the component repository, that is, deploying artifacts. If everything went well until now, nothing breaks, and no quality gates are missed, we can now deploy the previously created artifact to the component repository.

What has to be done? First of all, we add a shell build step to the newly created Hudson build job for Stage 3. To be more consistent and to document what's going on, the build step copies the previously created artifact to the workspace local to this build job, as shown in **Listing 6**.

Figure 5

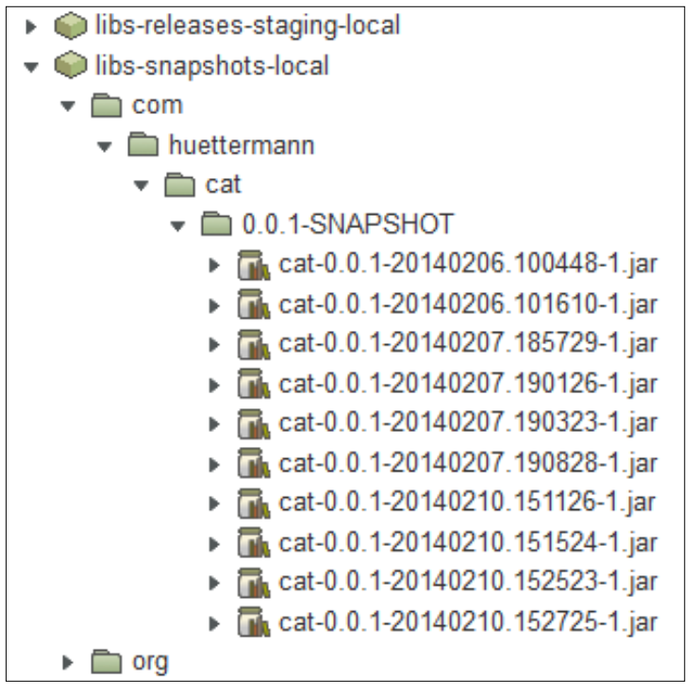


Figure 6

Now we need to configure the component repository, in our case, Artifactory. Artifactory is a component repository that can be smartly integrated with Hudson. It serves as a tool for artifact exchange and provisioning across all artifact types, including Java, Linux RPMs, and many more.

In the Hudson build job, we now configure the desired Artifactory target repository—that is, a logical target path in the overall repository. Side note: having the Git baseline as well, it would be easy to just rebuild the Maven project and deploy the outcome directly to Artifactory. In that case, we would not use mvn clean deploy because that is an antipattern. The better approach would be to fire an mvn clean install command and let Artifactory do the deployment in a transactional way.

LISTING 6 LISTING 7

```
#!/bin/sh
rm *.jar
cp /home/michael/talk/transfer/*.jar .
```

Download all listings in this issue as text

But, in our case, requirements might differ, for instance, needs for highly performant build cycles, and we've decided to take the built artifact from Stage 1 and deploy it with Artifactory's generic integration facility. For this, we can use an Ant-style configuration of the file pattern for the published artifacts (what to deploy) and target destination (where to deploy). Our configuration might look like **Listing 7**.

Now we're done with deploying ongoing snapshot artifacts to Artifactory as part of our continuous build. Our Hudson build job history shows the Artifactory icon to document that binaries were deployed to Artifactory; see **Figure 5**, which shows that the Hudson build job history visualizes that artifacts were deployed to Artifactory.

It comes in handy to be able to directly navigate from the build history to the built artifacts by just selecting the Artifactory icon of a specific build. We're then directed to Artifactory and can work on or just obtain information for those artifacts. **Figure 6** shows the Artifactory browser listing different deployed snapshot versions. As you see, although we deploy snapshot versions, inside Artifactory, artifacts are handled as unique, fully qualified versions. This is achieved by pure configuration inside Artifactory.

In Artifactory, artifacts are browsable. You can operate on those artifacts depending on your permissions.

We're done: Git commits continuously trigger the pipeline, and new snapshot versions are produced and distributed. That's fine for development, but we move on in the releasing process and sooner or later (with luck, sooner), we want to create a release, which we explore next.

Creating a Release

Our releasing process is implemented in a dedicated Hudson build job. First, we want to check out a previously compiled and tested baseline of the continuous build. Thus, we parameterize the Hudson release build job with an input field of type String and assign HEAD to be its default value. During build job execution, it's highly recommended to enter a specific, well-tested Git commit hash, but for build chain testing purposes, HEAD might be sufficient. As the first build step, we can then access the parameter and check out the given Git commit as a shell execution by using git checkout $rev.

Hudson's Git plugin creates a clone, and performing the checkout aligns us with the desired baseline. Maven's Release plugin is often a good choice—see Section 5.4 in my *Agile ALM* book (Manning, 2011)—but sometimes it's not the best fit for the given requirements. An alternative is to directly apply fine-grained steps as needed. Examples include tagging in the VCS (for example, by executing shell commands as build steps) or setting the version number in the POM files. The latter can be achieved easily by a self-written Maven Mojo, as shown in **Listing 8**.

The special trick with this Maven plugin is that it reads the project version of the Maven project it was applied on, strips its snapshot version, and dynamically assigns the result (that is, the release version) to the property newVersion. That property, in turn, is the input of the set goal of the Maven Versions plugin. Thus, in the Hudson build job, we can fire the command shown in **Listing 9** on the underlying to-be-released Maven-based project.

We skip the generation of backup POMs, because we work on defined baselines and we can always roll back. Even better, we always roll forward because we just work on the new changes piped through the pipeline. Also, here, it does not make any difference whether we work on one POM or any hundreds of child modules. The procedure is the same.

After we've patched the POM, the source code on the build machine is now ready to be tagged and deployed to a full-fledged component repository. We use mvn clean install to install the release artifact locally. Afterward, depending on our requirements, we orchestrate the target version number, that is, the version number the artifact is labeled with in the component repository. For that, we execute the shell as a build step in our Hudson release build job (see **Listing 10**).

After processing, the release variable contains a string that is enriched with a lot of useful context information including the version from the POM file, the first six characters of the Git hash, the Git revision number, and the build number of the executing Hudson build job. It's just an example, but you get the point: take the information that is most helpful to you; concrete requirements might influ-

LISTING 8 LISTING 9 / LISTING 10

```
package VersionFetcher;

import org.apache.maven.plugin.AbstractMojo;
import org.apache.maven.plugin.MojoExecutionException;
import org.apache.maven.project.MavenProject;


/**
 * @goal release
 * @phase process-sources
 */
public class VersionFetcher extends AbstractMojo {

  /**
   * @parameter expression = "${project}"
   * @readonly
   */
  private MavenProject project;

  public void execute() throws MojoExecutionException {
  String version = project.getVersion();
  String release = version;
  if (version.indexOf("-SNAPSHOT") > -1) {
    release = version.substring(0, version.indexOf(
"-SNAPSHOT"));
    getLog().info("SNAPSHOT found: " + release);
  }
  project.getProperties().setProperty("newVersion", release);
 }
}
```

Download all listings in this issue as text

ence your choice. But be aware of any implications. For example, only adding the build number from the Hudson build job itself is too fragile, because it is not unique across different Hudson build jobs.

Let's now proceed with our releasing process. The local target directory contains the release artifact that was built by Maven. We want to assign our individual release version to it; thus, we copy the artifact, as shown in **Listing 11**.

Now let's deploy the release artifact to our component repository. In our case, we use Bintray. Bintray is like a "GitHub for binaries." It's a social media service for developers to publish, download, store, promote, and share open source software packages. We use Bintray's powerful REST API to distribute our artifact.

First, for testing purposes only, we delete the target version of our component (in Bintray notation, this is a "package") in Bintray, as shown in **Listing 12**. This comes in handy if the version is available already and we don't need it.

As you see, a cURL command is sufficient, but you can, of course, also wrap the handling with Groovy (see this library for encapsulating the access) or any other language of choice.

Bintray follows some conventions. We host repositories (in our case, meow), which is owned by me, michaelhuettermann. An access key must be generated in Bintray, and we deposit it in Hudson as a configuration variable. Our repository consists of a package named cat.

Next we create the new target version by defining a Hudson build step to execute a shell script, as shown in **Listing 13**. After we've created the new version, we're ready to deploy the binary, as shown in **Listing 14**.

Finally, we have to make the binary visible; that is, we have to publish it. We can do that either in Bintray's web application itself or we can again execute an API call (see **Listing 15**).

As a result, we now have distributed the artifact, in its new version, to Bintray. **Figure 7** shows that the freshly published versions of artifacts are available in Bintray.

Crisp, isn't it?

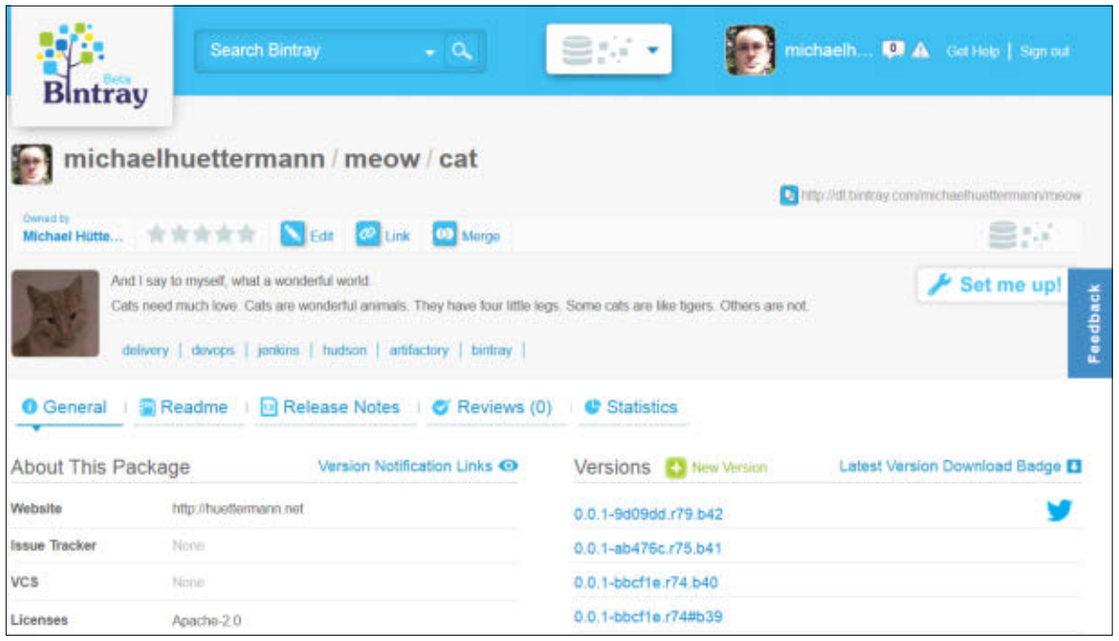


Figure 7

LISTING 11 / LISTING 12 / LISTING 13 / LISTING 14 / LISTING 15

```
cp devops/target/cat-$version.jar devops/target/cat-$release.jar
```

Download all listings in this issue as text

Conclusion

This article explained some concepts for mastering binaries with Maven projects, and it introduced a powerful tool chain. We've done a round-trip through some concrete examples to show how part of a delivery pipeline can look. Now you are prepared to further streamline your software delivery process. As a result, you'll have more effective and efficient processes and tool chains and even more fun with your daily work.

Now, I wish you much success with mastering your binaries and fun trying out these tools yourself! `</article>`

MORE ON TOPIC:

Development Tools and Techniques

LEARN MORE

- *DevOps for Developers*, by Michael Hüttermann (Apress, 2012)
- *Agile ALM*, by Michael Hüttermann (Manning, 2011)

STEPHEN CHIN

BIO

Mary Had a Little Lambda

Get familiar with lambdas and the Stream API through a simple game.

Lambda expressions are the most impactful feature to enter the Java language since the release of generics in Java SE 5. They fundamentally change the programming model, allowing a functional style of development, and they support efficient parallelization of code to take advantage of multicore systems. However, as a Java developer, you will first notice the productivity improvements you gain by using the new lambda-enabled APIs in Java SE 8.

In this article, we will use a retro game written in JavaFX to walk through the new Stream API for working with collections and data. This game is a simple Java SE 8 application written from the ground up to showcase lambdas best practices, and it is also a visual guide to programming with the Stream API. However, we will first lay the foundation with an introduction to the lambdas language changes.

Introduction to Lambdas

To use lambdas, you must be using a recent Java SDK (version 8 or higher) and set the language level to Java SE 8 when you compile. You can download the latest Java SDK version here.

Developing lambdas is a lot easier when using an IDE that supports the new syntax. Most Java IDEs have been updated with lambdas support and will assist you with real-time error reporting and code completion of lambdas. NetBeans IDE and IntelliJ are noteworthy as having the best lambdas support out of the box at the time of the Java SE 8 release, and both work well with the example we are demonstrating here.

To demonstrate how the new lambdas feature works, here is a short snippet of code that iterates through a list of shapes and changes the blue ones to red:

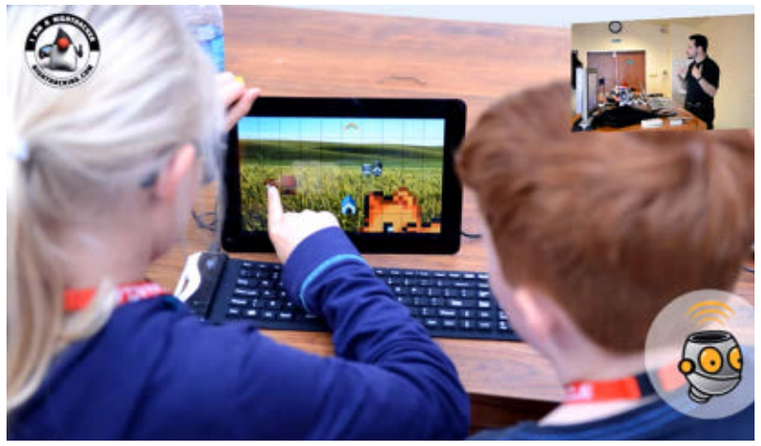
```
for (Shape s : shapes) {
  if (s.getColor() == BLUE)
    s.setColor(RED);
}
```

In Java SE 8, you could rewrite the same code by using a forEach and a lambda expression, as follows:

```
shapes.forEach(s -> {
  if (s.getColor() == BLUE)
    s.setColor(RED);
});
```

The lambda form makes use of a new method on the Collection interface called forEach, which takes a lambda expression and evaluates it for all the contained elements. Similar API enhancements have been made throughout the Java core classes in order to simplify the usage of lambda expressions.

A related question you might have is how the Java team was able to add in new methods to interfaces without breaking backward



At a Devoxx4Kids session, kids learned to code using the sample application in this article.

compatibility. For example, if you have code that implements the Collection interface and does not have a forEach method defined, won't the upgrade to Java SE 8 break your implementation? Fortunately, another feature called *extension methods* solves this problem in Java SE 8. The implementation of forEach on the Collection interface is shown in **Listing 1**.

Notice the new default keyword, which indicates that the method will be followed by a default implementation. Subclasses are free to create their own implementation of the method, but if none is defined, the subclasses will get the standard behavior defined in the interface. This allows new methods to be added to existing interfaces in the core Java classes, as well as in your own libraries and projects.

The actual lambda syntax is quite simple: in its full form, you supply the types and parameters on the left, insert a dash followed by the greater-than sign (->) in the middle, and follow that with a method body inside curly braces, as shown below:

```
(int a, int b) -> { return a + b; }
```

In cases where the function returns a value, the code can be simplified by removing the curly braces, the return keyword, and the semicolon:

```
(a, b) -> a + b
```

Furthermore, in cases where there is only one parameter, you can leave off the parentheses:

```
a -> a * a
```

And finally, if you have no parameters, you can simply leave the parentheses empty, as shown below, which is common for replacing Runnable implementations or other no-parameter methods.

```
() ->{ System.out.println("done"); }
```

METHOD REFERENCE	LAMBDA EQUIVALENT	
Objects::toString	obj -> Objects.toString(obj)	STATIC METHOD REFERENCE
Object::toString	obj -> obj.toString()	MEMBER METHOD REFERENCE
obj::toString	() -> obj.toString()	OBJECT METHOD REFERENCE
Object::new	() -> new Object()	CONSTRUCTOR METHOD REFERENCE

Table 1

LISTING 1

```
interface Collection<T> {
  default void forEach(Block<T> action) {
    Objects.requireNonNull(action);
    for (T t : this)
      action.apply(t);
  }
  // Rest of Collection methods...
}
```

Download all listings in this issue as text

In addition to the basic syntax, there is also a special shortcut syntax called *method references*, which lets you quickly create lambda expressions that refer to a single method as the implementation. The following table summarizes the different types of method references along with the equivalent long-form lambda syntax.

The last concept that is important when working with the new lambdas methods is the creation of interfaces that allow you to accept lambda expressions. For this purpose, any interface that has one explicitly declared abstract method can be used to accept a lambda expression and is, thus, called a *functional interface*.

As a convenience, a new FunctionalInterface annotation was introduced that optionally can be used to mark interfaces in order to get assistance from the compiler in checking that the interface meets the requirement for a single, explicitly declared abstract method:

```
@FunctionalInterface
interface Sum {
  int add(int a, int b);
}
```

Using this annotation is a recommended best practice, because it will catch corner cases in the definition of functional interfaces—such as the inclusion of default methods that allow you to have multiple methods defined on a functional interface, because they are not abstract and don't count toward the single-abstract-method requirement.

Now that you have a basic understanding of the lambda syntax, it is time to explore the Stream API and see the power of lambdas in the context of a visual example.

Retro Gaming with Lambdas

Mary had a little lambda
Whose fleece was white as snow
And everywhere that Mary went
Lambda was sure to go!

Nowadays video games are all about high-resolution 3-D graphics, cinematic-quality cut scenes, and difficulty levels that range from newbie to pacifist. However, in the good old days of gaming, we just had sprites (see **Figure 1**): cute, pixelated little figures dancing and walking their role-playing game (RPG)–way through well-designed and insanely difficult levels.

Figure 1

Figure 2

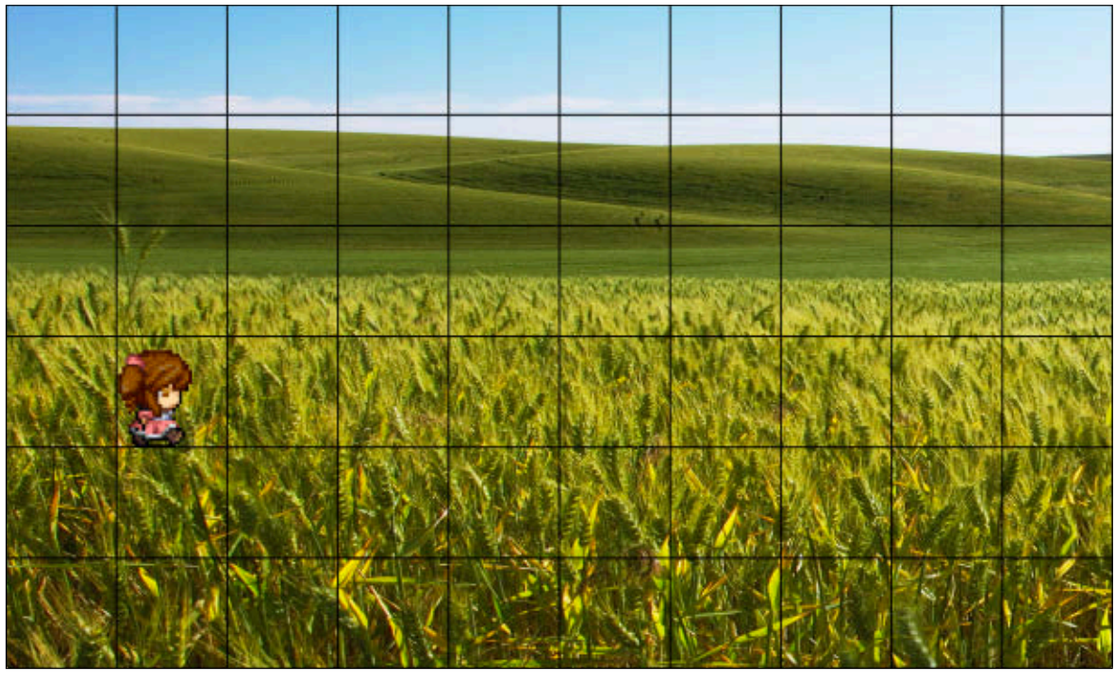


Figure 3

LISTING 2 LISTING 3

```
ChangeListener<Object> updateImage =
 (ov, o, o2) -> imageView.setViewport(
  new Rectangle2D(frame.get() * spriteWidth,
            direction.get().getOffset() * spriteHeight,
            spriteWidth, spriteHeight));
direction.addListener(updateImage);
frame.addListener(updateImage);
```

Download all listings in this issue as text

Sprite-based graphics also happen to be really simple to program, allowing us to build a full animation system in under 400 lines of code. The full application code for our game is in GitHub. For all the graphics used in the game, the images are laid out in a standard 3 x 4 tiled format, as shown in the sprite sheet for Mary (see **Figure 2**).

The code for animating sprites is done (of course) using a lambda, and it simply moves the viewport around a tiled image in order to produce a three-frame walking animation (horizontal) and to change the direction the character is facing (vertical). See **Listing 2**.

Add a static image for a background (see **Figure 3**), and some key event listeners to move the character upon input, and you have the basics of a classic RPG game.

Generating Streams

There are several ways to create a new Java SE 8 stream. The easiest way is to start with a collection of your choice and simply call the stream() or parallelStream() method to get back a Stream object, such as in the following code snippet:

```
anyCollection.stream();
```

You can also return a stream from a known set of objects by using the static helper methods on the Stream class. For example, to get back a stream that contains a set of Strings, you could use the code in **Listing 3**.

Similarly, you can use the Stream numeric subclasses, such as IntStream, to get back a generated series of numbers:

```
IntStream.range(0, 50)
```

But the most interesting way to generate a new series is to use the generate and iterate methods on the Stream class, which let you create a new stream of objects using

Figure 4

LISTING 4 LISTING 5 / LISTING 6

```
Stream.iterate(Color.RED,
  c -> Color.hsb(c.getHue() + .1, c.getSaturation(),
                 c.getBrightness()));
```

Download all listings in this issue as text

a lambda that gets called to return a new object. The iterate method is of interest because it will pass in the previously created object to the lambda. This lets you return a distinct object for each call, such as returning all the colors in the rainbow iteratively (see **Listing 4**).

To demonstrate how this works visually, we are going to add a new barn element to the application, which generates a lamb when Mary steps on it. The code for the new Barn class is shown in **Listing 5**. This code specifies the image to use for the sprite-based graphic, which is passed in to the superconstructor, and implements a visit method that has the logic that will get executed when Mary steps on the barn.

The first statement in the visit method simply gets the last element from the list of animals that are following Mary, or it returns her if there are no animals yet. This is then used as the seed to the iterate method, which gets passed to the Lamb constructor for the first invocation of the lambda. The lamb that gets generated by this is then passed in to the Lamb constructor for the second invocation, and this process repeats in succession.

The resulting stream includes the seed (so we can use the skip function to remove that from the stream), and it is theoretically infinite. Because streams are lazy, we don't need to worry about objects getting created until we add a terminal operation, but an easy way to fix the length of the stream is to use the limit function, which we will pass a parameter value of 7 to generate seven lambs that are following Mary.

The last step is to add a terminal operation that will use the stream. In this case, we will use a forEach function that applies a method reference to the add method on the list of animals. The result of executing this lambda is the addition of sevens lambs following Mary in succession, as shown in **Figure 4**.

The next element we are going to add to the game is a rainbow that will demonstrate filtering in the Stream API. The way the filter function works is that it takes a predicate lambda, which evaluates to true or false for each element in the stream. The resulting stream contains all the elements for which the predicate lambda evaluated to true.

For the logic of the rainbow, we will execute a filter that returns every fourth animal in the stream and apply a JavaFX ColorAdjust function to shift the hue to match the passed-in color. For white, we are using null (no color shift). The code in **Listing 6** is the implementation of the visit method for the rainbow MapObject.

When Mary steps on the rainbow, all the lambs get colored according to the Color values you

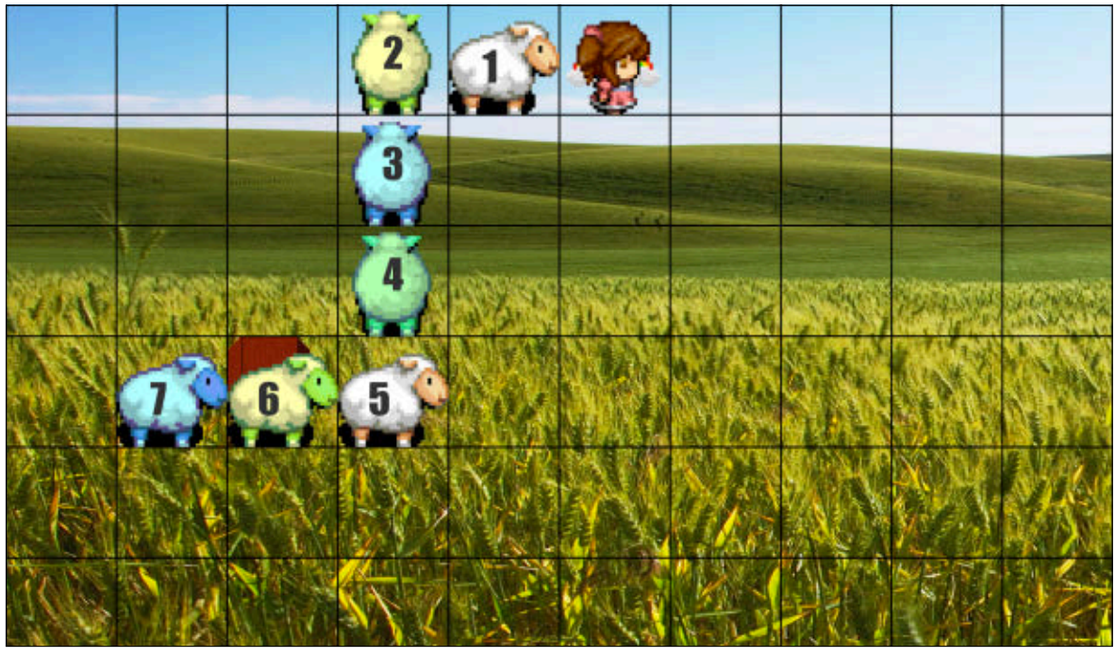


Figure 5

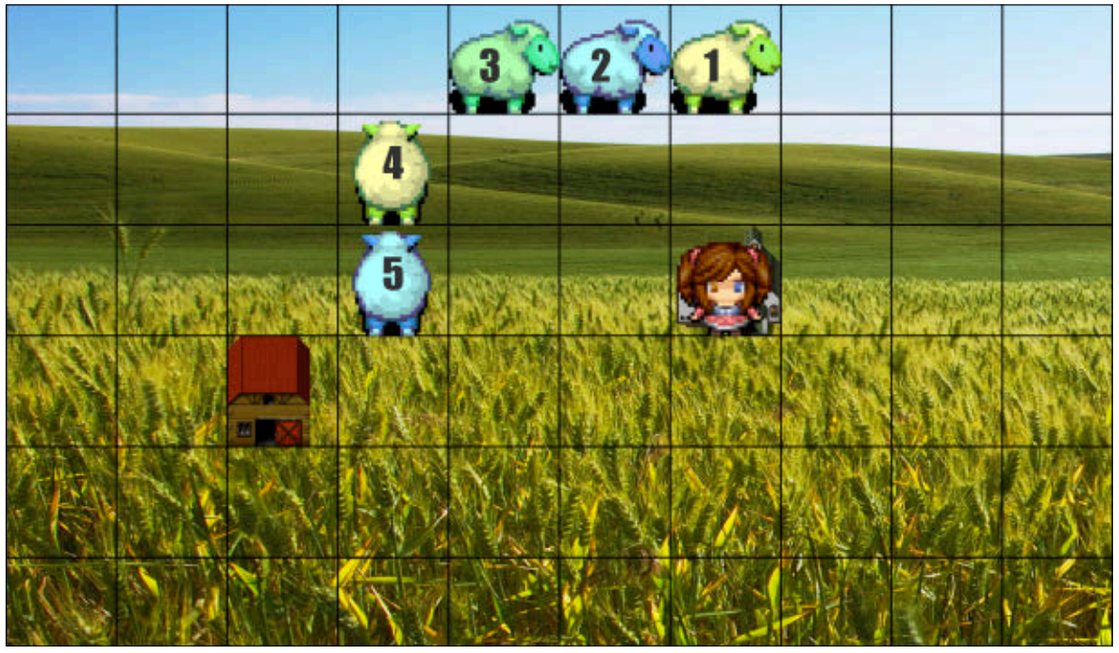


Figure 6

specified (see **Figure 5**).

"Lamb"da question 1. What happens if you step on the barn after visiting the rainbow?

Another way to use filtering is to take advantage of the new methods added to the Collection API that accept a predicate lambda.

LISTING 7 LISTING 8

```
Predicate<SpriteView> pure =
  a -> a.getColor() == null;

mealsServed.set(mealsServed.get() +
  s.getAnimals().filtered(pure).size()
);

s.getAnimals().removeIf(pure);
```

Download all listings in this issue as text

These include removeIf, which filters out all the elements that don't match the given predicate, and filtered, which is on ObservableList and returns a FilteredList containing only the items that match the predicate.

We will use these methods to implement a Church object that will filter on "pure" (white) animals; then, any animals that are white will be cooked by the church staff to feed the needy. This functionality includes incrementing the counter of "meals served" and removing the "pure" animals from the list. The code for the church visit method is shown in **Listing 7**.

You can see the result of successively stepping on the rainbow and the church in **Figure 6**.

"Lamb"da question 2. Is it possible to use the church to remove all the animals after they have already been colored?

The map function is probably the most powerful operation in the Stream API. It allows you to convert all the elements in the stream from one type of object to another, performing powerful transformations along the way. We will use this to implement a chicken coop where all the animals following Mary will get converted into eggs.

I have two implementations of the visit method for the chicken coop. The first one uses a single map operation with a lambda expression to replace the stream elements with eggs, as shown in **Listing 8**.

The second implementation uses method references with a chained set of map operations to first convert the stream to a stream containing who the animals are following, and then to call a constructor method reference to create the eggs, passing in the information shown in **Listing 9** to the constructor parameter.

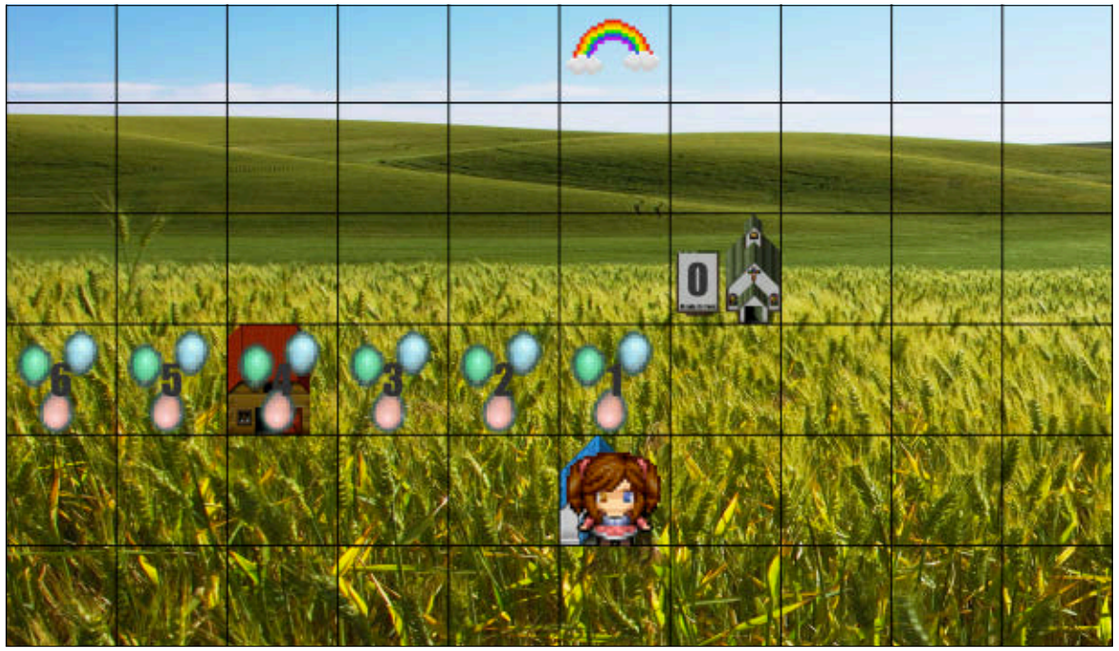


Figure 7

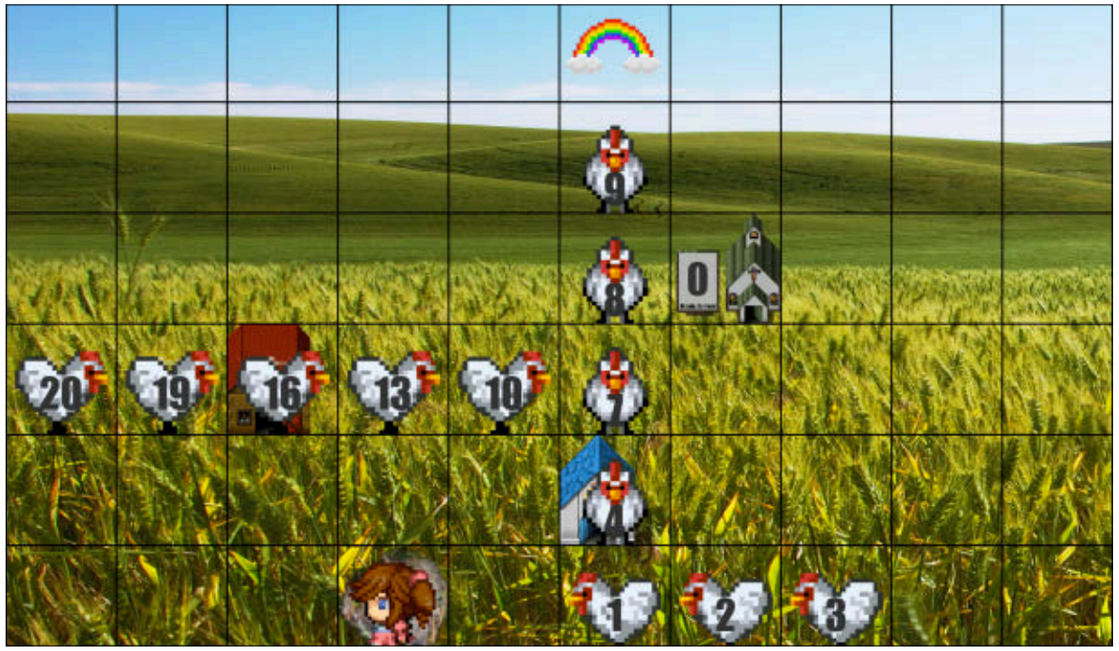


Figure 8

The code in both of these listings behaves and performs similarly, because the Stream API is designed to be lazy and only evaluate the stream when a terminal operation (such as collect) is called. Therefore, it is primarily a style issue as to which version you prefer to use.

LISTING 9 LISTING 10 / LISTING 11 / LISTING 12

```
// or a double map:
s.getAnimals().setAll(s.getAnimals()
   .stream().parallel()
   .map(SpriteView::getFollowing)
   .map(Eggs::new)
   .collect(Collectors.toList())
);
```

Download all listings in this issue as text

Running the program with the new chicken coop MapObject will let you generate eggs from lambs, as shown in **Figure 7**.

"Lamb"da question 3. If you send colored lambs to the chicken coop, what color are the eggs?

Notice that each of the egg sprites contains three little bouncing eggs. Wouldn't it be nice if we could hatch these guys into chickens?

To hatch the eggs, we will add in a new MapObject for a nest where the eggs will be hatched into a group of three chickens using the hatch method shown in **Listing 10**.

Notice that the hatch method returns a stream of objects, which means if we used a normal map operation we would get back a stream of streams. To flatten the stream into a single list of chickens, we can instead use flatMap, which will map the stream using a lambda function and also collapse the nested streams into a single list of objects. The implementation of the nest visit function utilizing flatMap is shown in **Listing 11**.

Now, upon bringing eggs to the nest, you will get an explosion of chickens, as shown in **Figure 8**.

"Lamb"da question 4. About how many animals can you add before the game runs out of memory?

The final element we will add is a fox to demonstrate how to reduce a stream. For this, we will first map the stream to a list of integers according to the weight of the animals, and then we will reduce that to a single value using a sum method reference. See **Listing 12**. The reduce function takes a seed value (for which we will use 0) and a function that can reduce two elements into a single result. This lambda will be applied recursively for all the elements in the stream until a single value results, which will be the sum of all the animals' weights.

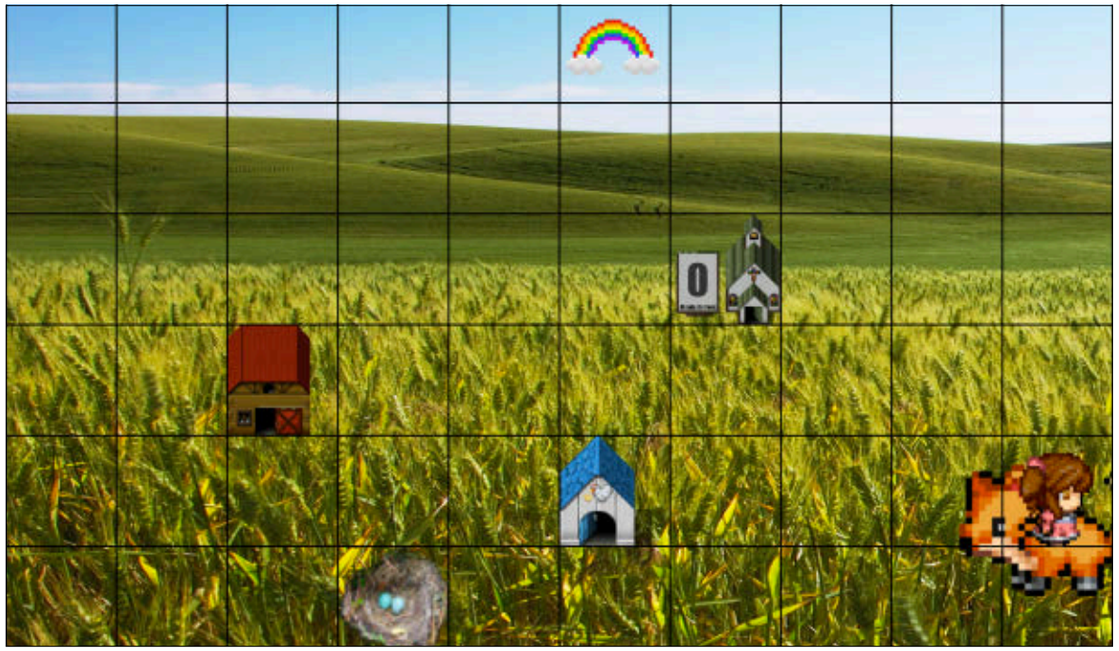


Figure 9

We then take the sum (stored into the variable called mealSize) and use that to stretch the fox proportionally. **Figure 9** shows the result of a tasty meal for the fox.

"Lamb"da question 5. How can you change the code for the fox to make him fatter when he eats?

Conclusion

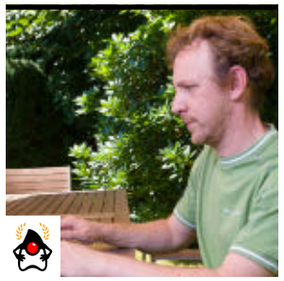
In this article, we covered the basic lambda syntax, including method references, extension methods, and functional interfaces. Then we went into detail about the Stream API, showcasing some of the common operations, such as iterate, filter, map, flatMap, and reduce.

As you have seen, Java SE 8 lambdas dramatically shift the programming model—allowing you to write simpler and more-elegant code, and opening up the possibility of new powerful APIs such as Stream. Lambdas are also used extensively throughout the Java core APIs, including new functions on I/O, inclusion in the new Date and Time API, and functional interfaces added to APIs with callbacks like JavaFX. This all leads to a more functional style of programming that you can use to build code that runs efficiently on multiprocessor systems. Why not start taking advantage of these capabilities in your own code? `</article>`

LEARN MORE

- Stream API
- Example game on GitHub

CREATE
THE FUTURE
oracle.com/java
Java™
ORACLE



JOHAN VOS

BIO

Leap Motion and JavaFX

Use 3-D hand movements to interact with JavaFX applications.

Sooner or later, there might come a day when parents have to explain to their children what a keyboard is. Children will have a hard time understanding why their parents used such a strange device for many decades. Although today is not yet that day, we are already seeing devices that provide more-intuitive, natural input than a keyboard, a mouse, or a trackpad do. These devices do not necessarily replace existing input devices, but they can be very complementary.

It is important that the software we write and use today not be restricted to a limited set of input devices. The JavaFX platform can easily be integrated with new input devices. This article shows how the Leap Motion Controller device can be used as a user input device in JavaFX applications.

The Leap Motion Controller, a small device produced by the Leap Motion company, is equipped with infrared cameras and infrared LEDs. This device is capable of tracking hand and finger movements around the device in a clipped pyramid that has a radius of about 1 meter. The device is connected to a computer using a USB cable. Leap Motion provides native drivers for Windows, Linux, and Mac systems. On top of those drivers, a number of APIs for different programming languages are available. These APIs allow developers to create applications that leverage the Leap Motion Controller.

Fortunately, a Java API is available. We will now explore how you can use the Java API in a Java client application to interact with the Leap Motion Controller, but first we will briefly discuss the capabilities of the Leap Motion Controller.

For a thorough understanding of the Leap Motion Controller, see the Leap Motion website and the Leap Motion developer website. Also visit the developer website to download the Leap Motion Software Development Kit (SDK), which allows you to create applications.

Johan Vos demonstrates how to use a Leap Motion Controller with a JavaFX application.

PHOTOGRAPH BY TON HENDRIKS

Using the Leap Motion SDK

Leap Motion provides an SDK for Windows, Mac OS, and Linux. The SDK contains a number of files, some of which are required for Java development, for example:

- A Java library named LeapJava.jar, which provides the API we can use in Java applications
- Operating system–dependent native libraries that connect to the Leap Motion Service, which receives data from the Leap Motion Controller device over USB

To run Java applications that communicate with the Leap Motion Controller, the native libraries need to be in the native library path. This is

achieved by starting the Java applications with the system property java .library.path pointing to the location of the native libraries for your particular OS, for example, java -Djava.library.path=/path/to/LeapSDK/lib/x64 on a 64-bit system.

The Java library LeapJava.jar file should be in the classpath.

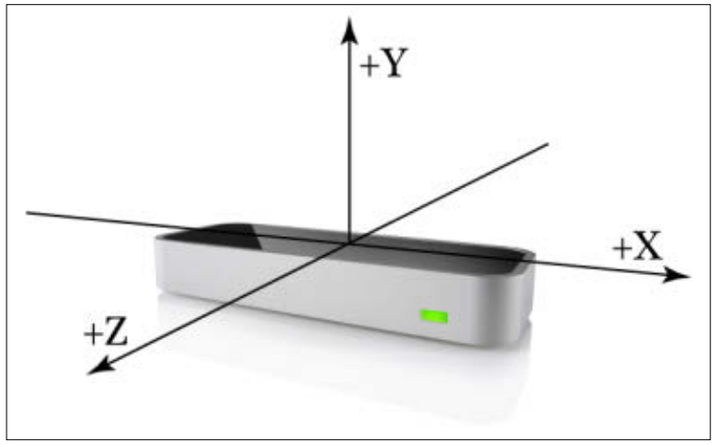


Figure 1

Leap Motion Controller Concepts

Because the Leap Motion Controller is capable of tracking hands and fingers in three directions, the retrieved data is obtained in a three-dimensional right-handed Cartesian coordinate system. The origin of this coordinate system is located at the center of the top of the device. The *x*, *y*, and *z* axes are pointing in the directions shown in **Figure 1**.

All data obtained from the Leap Motion Controller is contained in instances of the com.leapmotion .leap.Frame class, which is part of the LeapJava.jar library. Depending on environment variables, the device sends frame data with a frequency between 30 Hz and 200 Hz.

There are two ways of obtaining the frame data, both of which require an instance of the com .leapmotion.leap.Controller class to operate:

- By polling the Controller
- By registering a Listener with the Controller, which will be notified when new data is available

When using JavaFX, the second approach is the most intuitive. The callback function on the Listener can be used to change properties in the data model of the application, and the JavaFX pulse thread will make sure the changes are reflected in the user interface. Because a pulse event is generated at most 60 times a second, the UI thread won't be overloaded when the Leap Motion Controller provides more data than the graphical system can handle.

The code in **Listing 1** shows how to create a Controller and register a Listener in a JavaFX application. The LeapListener class we introduce here extends the com.leapmotion .leap.Listener class, as shown in **Listing 2**.

LISTING 1 LISTING 2

```
public class LeapConcepts extends Application {
  Controller c;
  Listener l;

  public void start (Stage stage) {
     .... // set up the scene and stage
    c = new Controller();
    l = new LeapListener(this);
    c.addListener (l);
  }
}
```

Download all listings in this issue as text

This Listener class will be called upon lifecycle events and whenever a frame containing tracked data is available. The onConnect method is called whenever the controller instance connects to a Leap Motion device. When, for some reason, the connection is broken, the onDisconnect method is called. Obviously, the onFrame method is called whenever a new frame with data is available.

Available Data

As mentioned before, all information we receive from the Leap Motion Controller is available on instances of the Frame class. This article gives a brief, nonexhaustive overview of the data that is available. The Java documentation on the Leap Motion website contains all the information for this class and the other classes.

The Leap Motion Controller is capable of tracking the position, direction, and movement of hands, fingers, and tools (for example, a pencil). Based on internal calculations, this allows the Leap Motion Service to also track a number of gestures, for example, swiping, tapping, and making a circular move.

If we want information on the tracked hands in a frame, we call frame.hands(). This method returns a HandList, which is an Iterable that can be used to obtain a number of tracked Hand instances. A Hand instance has a number of properties: for example, the location, the normal vector, and the velocity of

the hand's palm or the fingers that are detected on the hand.

Fingers can be tracked individually as well by calling frame.fingers(). This method returns a FingerList that can be used to obtain information about the detected fingers (for example, direction, location, and velocity).

Consequently, gestures are obtained by calling frame.gestures(), which returns a GestureList. Note that gestures will be added to the frame data only if the Controller is instructed to do so. Hence, it is good practice to enable gesture tracking in the onConnect method of the class extending the Listener class, as shown in **Listing 3**.

The code in **Listing 3** will make sure that when swipe or key-tap gestures are performed by the user, the corresponding information is added to the Frame instances that are provided.

Threading

Both the JavaFX platform and the Leap Motion Controller software have some specific requirements regarding threading. Fortunately, these requirements match very well.

When writing JavaFX applications, it is good practice to separate the view and the data. That is, user interface components are defined and glued together, but the data that describes their behavior (for example, the position of a circle or the width of a text field) is kept in JavaFX properties. These properties are altered whenever needed, but this does not mean that the user interface is redrawn every time a property changes.

It is a requirement, though, that changes to properties that can lead to changes in the user interface be executed on the JavaFX application thread.

The Leap Motion native library will create a new Java thread whenever a new Frame instance is available, and it will call Listener .onFrame()—if a Listener is registered with the Controller, that is.

The rate by which new threads are created is between 30 and 200 times a second. However, it turns out that a new thread is created only when the previous thread completed its work, that is, when the Listener.onFrame() method returns. This prevents a thread-flooding situation, where threads are created at a higher pace than they can be processed.

The combination of these two systems leads to an approach where the onFrame() method on the Listener is used to (directly or indirectly) change the properties of UI components, which are subsequently rendered. The schema shown in **Figure 2** describes the flow.

In this flow, the implementation of the Leap Motion Listener is using the Platform.runLater() approach to change properties of the JavaFX controls. This guarantees that those properties are modified only by the JavaFX application thread, as required by the JavaFX platform.

In practice, and especially in more-complex applications, it is often useful to add a step between the Listener implementation and the JavaFX controls. By doing so, the separation between the Leap Motion interface and your JavaFX application is even clearer.

For brevity, in this article, we will use the approach where the implementation of the Listener modifies the properties of the JavaFX controls directly, using the Platform .runLater() approach.

We will demonstrate the flow

LISTING 3

```
@Override
public void onConnect (Controller c) {
  c.enableGesture(Gesture.TYPE.TYPE_SWIPE);
  c.enableGesture(Gesture.TYPE.TYPE_KEY_TAP);
}
```

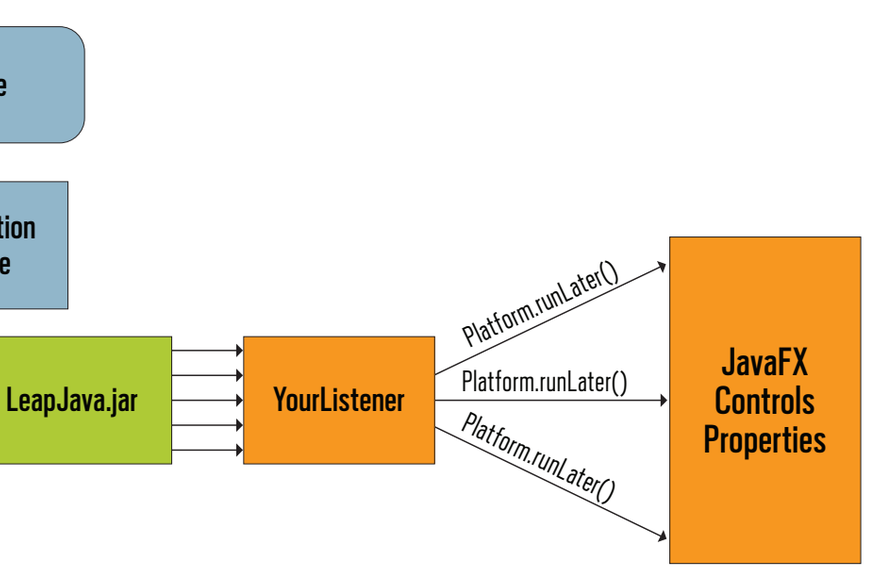

Download all listings in this issue as text

Figure 2

with a very simple example: we will write a JavaFX application that displays a circle. The location and the radius of the circle are determined by hand movements. The full code for this example is available at this Bitbucket site.

As stated at the beginning of this article, JavaFX allows you to create client applications in which there is a large degree of independence between the layout and the input devices. We will first create a JavaFX application that is not dependent on the presence of the Leap Motion Controller. The code in **Listing 4** generates a layout with a circle positioned in the middle.

As you can see from this code, the location (via the translateX and translateY properties) and the radius (via the radius property) are bound to JavaFX properties. These properties are declared in our JavaFX application, and they are made available via public methods, as shown in **Listing 5**.

So far, this application is very static. It shows a green circle with a fixed radius at a fixed position. We will now write a class that listens for Leap Motion Controller data. We do this by extending the Leap Motion Listener class, as shown in **Listing 6**.

In this class, we override the onFrame() method. Our implementation of this method will be called when new frame data is available. The frame data is obtained by calling Frame frame = controller.frame();, where the controller instance is passed via the method invocation.

As shown in **Listing 6**, we can obtain the detected hands easily by calling HandList hands = frame .hands();. If no hands are detected, the hands.isEmpty() call will return true, and we do nothing. If at least one hand is detected, we obtain it by calling Hand hand = hands .get(0);. The position of the palm of the detected hand is obtained as a three-dimensional vector by calling hand.palmPosition().

DID YOU KNOW?
JavaFX allows you to create client applications in which there is a **large degree of independence** between the layout and the input devices.

We map the *x* and the *z* coordinates of the Leap Motion coordinate system with the translateX and translateY properties of the circle we created in the JavaFX application. The *y* coordinate is mapped to the radius property.

Note: Working with the Leap Motion APIs involves some mathematics. You have to transform the coordinates obtained from the Leap Motion Controller to pixels on the screen. The Leap Motion Controller coordinates are expressed in millimeters distance from the center of the top of the Leap Motion.

Because the onFrame method is called on a thread created by the native Leap Motion libraries, we cannot change JavaFX properties directly. Instead, we have to use the Platform.runLater() pattern in order to push the changes to the JavaFX properties onto the event queue.

The only remaining thing we have to do is to create an instance of our Listener and add it to a Controller. This is done in our application class, as shown in **Listing 7**.

The Controller is created on the JavaFX application thread, as is the Listener. These methods return immediately, though, and they do not freeze the user interface. It is important to maintain a reference to the controller object, in order to prevent it from being garbage collected.

LISTING 4 LISTING 5 / LISTING 6 / LISTING 7

```
public class LeapConcepts extends Application {
...
@Override
public void start (Stage primaryStage) {
    Circle circle = new Circle(20);
    circle.setFill(Color.GREEN);
    circle.translateXProperty().bind(centerX);
    circle.translateYProperty().bind(centerY);
    circle.radiusProperty().bind(radius);
    StackPane root = new StackPane();
    root.getChildren().add(circle);
    Scene scene = new Scene(root, 300, 250);
    primaryStage.setScene(scene);
    primaryStage.show();
}
...
}
```

Download all listings in this issue as text

Simple Map Application

The example we have explored so far is very basic, but it outlines

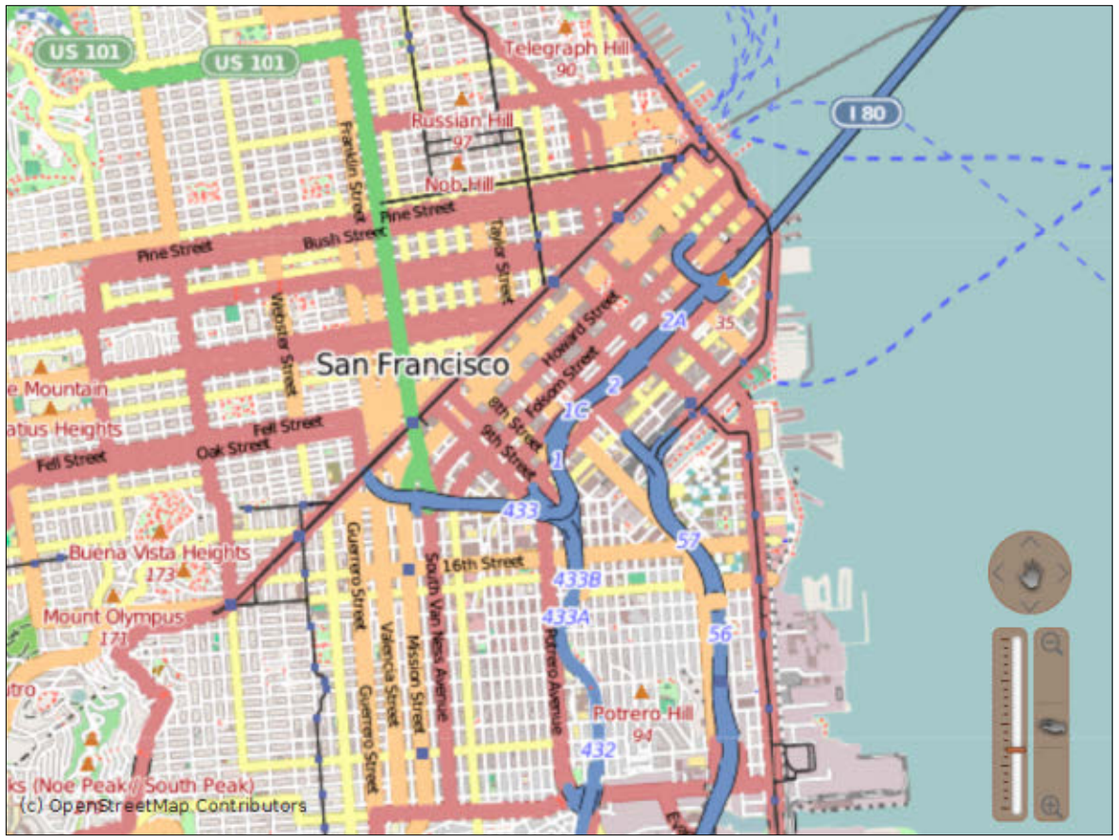


Figure 3

the core principles you have to use when integrating Leap Motion Controller data into your JavaFX application.

The possibilities are endless. We will close this article with a simple map application that is navigated using the Leap Motion Controller rather than with a mouse. The code for this map application is available here. **Figure 3** shows a screenshot of this application.

We apply the same core principle as in the previous example, which means that we first create a map application that does not have dependencies on the input device.

We create a MapArea instance that serves as the container where we put MapTile instances. A MapTile represents a 256-x-256-pixel piece of the world map at a given zoom level. We obtain these tiles from OpenStreetMap, which is an open data effort built by a community of mappers. The calculations required to map coordinates and zoom levels onto pixels is beyond the scope of this article. The MapArea class contains a loadTiles() method that will load new MapTile instances and their corresponding map images when needed. There are two potential causes for this:

- The user is panning the map, and a new area is shown
- The user is changing the zoom level, and a more detailed version of the map tiles should be rendered

All of this can be achieved via a traditional mouse or trackpad, but we can easily add a Leap Motion Listener that achieves the same end. We extend the Leap Motion Listener class and implement the onFrame method as shown in **Listing 8**.

LISTING 8

```
@Override
public void onFrame(Controller controller) {
  Frame frame = controller.frame();
  HandList handList = frame.hands();
  Iterator<Hand> handIterator = handList.iterator();
  while (handIterator.hasNext()) {
    Hand hand = handIterator.next();
    if (hand.isValid() && (hand.fingers().count() > 2)) {
      Vector palmPosition = hand.palmPosition();
      final float x = palmPosition.getX();
      final float z = palmPosition.getZ();
      final float y = palmPosition.getY();
      Platform.runLater(() -> {
        if (Math.abs(x) > 10) {
          area.moveX(x/10);
        }
        if (Math.abs(z) > 10) {
          area.moveY(z/10);
        }
        if (Math.abs(y-150) > 20) {
          area.moveZoom(y-150);
        }
      });
    }
  }
}
```

Download all listings in this issue as text

In this code, we once again detect the position of the hand. This time, however, we add an additional check: we will move or zoom only if the user opens his hand. We do this by asking how many fingers are detected. When a hand is closed, the Leap Motion Controller won't count fingers on it. As long as we count at least two fingers, we decide the hand has been opened.

Next, we will move the map area proportional to the location of the hand, and the zoom level will be changed proportional to the height of the hand.

USE YOUR INTUITION

Only a little bit of code is required to integrate the Leap Motion Controller into existing applications. There are huge challenges and, hence, huge opportunities for determining the best way to react to hand movements. Intuition is very important in this area.

Conclusion

As we showed in this article, only a little bit of code is required to integrate the Leap Motion Controller into existing applications. There are huge challenges and, hence, huge opportunities for determining the best way to react to hand movements. Intuition is very important in this area. Creative developers will probably have fun experimenting with this cool device.

The JavaFX platform provides a great environment for combining new input devices and existing or new Java code.

Special care is required when dealing with threading, but the JavaFX threading model allows for a perfect decoupling between the rendering of the user interface, and dealing with background events and computations.

Using JavaFX, developers can leverage the great graphical potential provided by the JavaFX controls. The whole Java platform is available for enriching the applications—for example, by providing communication to backend systems. `</article>`

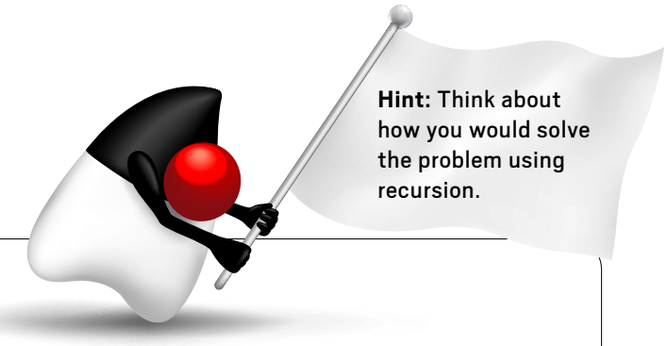
LEARN MORE

- Leap Motion website
- Leap Motion developer website
- NightHacking video: Interview with Johan Vos on JavaFX and Leap Motion

CREATE THE FUTURE

oracle.com/java

// fix this /



In the March/April 2014 issue, Romanian polyglot developer Attila Balazs gave us a concurrency challenge. He showed us code that sometimes mysteriously "drops" incrementation attempts and asked us for a fix. The correct answer is #3. The problem is that a race condition exists between the individual calls to cnts. For example, we can have a scenario where thread 1 reads the value for counter X and then thread 2 reads the same value. Both calculate the incremented value and both write it back, resulting in a +1 increment rather than a +2 increment as expected. There is a similar race condition with regard to initializing the counters.

Although #1 would work, it would defeat the purpose of using a ConcurrentHashMap. #2 wouldn't work because it has the same problems (individual operations are atomic rather than the complete sequence of events). #4 can be made to work, but it would require extra bookkeeping.

This issue's challenge comes from Simon Ritter, Java evangelist at Oracle, who presents us with a streams problem.

1 THE PROBLEM

Given code that uses the new Java SE 8 Stream API to determine the length of the longest line in a text file, how can we convert this to return the actual line, rather than its length?

2 THE CODE

The lines() method of the BufferedReader class is new in Java SE 8 and returns a stream of Strings that are the text lines of the file. This is passed to the mapToInt() method, which generates an IntStream using the method reference to String.length() as a function. The terminal operation max() identifies the largest value in the stream and returns an OptionalInt object. The get() method returns the value of the OptionalInt that is auto-unboxed to an int and assigned to the variable, longest.

```
BufferedReader reader = new BufferedReader(new FileReader
   ("foo.txt"));

int longest = reader.lines().mapToInt(String::length)
   .max().get();
```

How could we change this code to return the longest line in the file?

3 WHAT'S THE FIX?

1)
```
String longestLine = reader.lines().max(String::longest)
   .orElse("");
```

2)
```
String longestLine = reader.lines().reduce((a, b)
   -> a.length() > b.length() ? a : b).orElse("");
```

3)
```
String longestLine = reader.lines().reduce("", (a, b)
   -> a.length() > b.length());
```

4)
```
String longestLine = reader.lines().max((a, b)
   -> b.length() - a.length()).orElse("");
```

5)
```
String longestLine = reader.lines().sorted((a, b)
   -> a.length() - b.length()).findFirst()
   .orElse("");
```

GOT THE ANSWER?

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

PHOTOGRAPH BY BOB ADLER, ART BY I-HUA CHEN