

JavaTM magazine

By and for the Java community 

SEPTEMBER/OCTOBER 2015



KOTLIN 46 | FUNCTIONAL JAVA 50 | CDI 59 | CAREERS 77

Testing

HUNTING DOWN HARD-TO-FIND ERRORS

14 AUTOMATED
TESTING
FOR JAVAFX

20 JUNIT'S MOST
UNDERUSED
FEATURES

26 BUILDING
A SELENIUM
TEST GRID

14 TEST JAVAFX APPS WITH TESTFX

By Bennet Schulz

Simple JUnit-style testing of JavaFX UIs

20

EIGHT GREATLY UNDERUSED FEATURES OF JUNIT

By Mert Çalışkan

Make your testing a whole lot easier with little-used JUnit capabilities.

26

BUILDING AND AUTOMATING A FUNCTIONAL TEST GRID

By Neil Manvar

How to assemble a grid for Selenium testing

32

STRESS TESTING JAVA EE APPLICATIONS

By Adam Bien

Identify application server configuration problems, potential bottlenecks, synchronization bugs, and memory leaks in Java EE code.

42

THINK LIKE A TESTER AND GET RID OF QA

By Mark Hrynczak

Atlassian represents the bleeding edge in testing: its developers are required to formulate tests before they code and after. QA is there to help—but not test. Here's how it's working.

COVER ART BY I-HUA CHEN

03

From the Editor

One of the most effective tools for defect detection is rarely used due to old prejudices, except by companies who can't afford bugs. They all use it.

06

Letters to the Editor

Corrections, questions, and kudos

07

Events

Calendar of upcoming Java conferences and events

10

Java Books

Reviews of books on JavaFX and Java EE 7

46

JVM Languages

Kotlin: A Low-Ceremony, High-Integration Language

By Hadi Hariri

Work with a statically typed, low-ceremony language that provides first-class functions, nullability protections, and complete integration with existing Java libraries.

50

Functional Programming

Functional Programming in Java: Using Collections

By Venkat Subramaniam

Part 2 of our coverage of functional programming explains the code smells that can arise from lambda-based functional routines.

59

Java EE

Contexts and Dependency Injection: The New Java EE Toolbox

By Antonio Goncalves

More loose coupling with observers, interceptors, and decorators

70

Architecture

A First Look at Microservices

By Arun Gupta

The latest trend in enterprise computing is microservices. What exactly are they?

75

Fix This

Our latest code challenges from the Oracle certification exams

77

Career

More Ideas to Boost Your Developer Career

By Bruno Souza and Edson Yanaga

Skills to develop, activities to explore

25

Java Proposals of Interest

JEP 259: Stack-Walking API

45

User Groups

The Virtual JUG

80

Contact Us

Have a comment? Suggestion?

Want to submit an article proposal? Here's how to do it.



EDITORIAL

Editor in Chief

Andrew Binstock

Managing Editor

Claire Breen

Copy Editor

Karen Perkins

Section Development

Michelle Kovac

Technical Reviewers

Stephen Chin, Reza Rahman, Simon Ritter

DESIGN

Senior Creative Director

Francisco G Delgadillo

Design Director

Richard Merchán

Senior Designer

Arianna Pucherelli

Designer

Jaime Ferrand

Senior Production Manager

Sheila Brennan

Production Designer

Kathy Cygnarowicz

PUBLISHING

Publisher

Jennifer Hamilton +1.650.506.3794

Associate Publisher and Audience Development Director

Karin Kinnear +1.650.506.1985

Audience Development Manager

Jennifer Kurtz

ADVERTISING SALES

Josie Damian +1.626.396.9400 x 200

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)

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 © 2015, 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 at no cost to qualified subscribers by Oracle, 500 Oracle Parkway, MS OPL-3A, Redwood City, CA 94065-1600.

Digital Publishing by [GTxcel](#)



20 Years of Innovation

#1 Development Platform



Since 2002



Since 1999



Since 1995



Since 1996



Since 1998



Since 1996



Since 1996



Since 2008



Since 1998



Since 1999



Since 2001



Since 1996



Since 2012

ORACLE®

real life, of course, most organizations just call in consultants.)

Were you to perform this inquiry, it might lead to consideration of what practices you could add to existing projects to raise their overall quality without sacrificing productivity. In many—likely in most—cases, that additional practice would be static analysis of code. Relying on [the work of Capers Jones](#), one of the most widely experienced analysts of software-engineering data, static analysis reduces defects in projects by 44 percent (from an average of 5.0 defects per unit of functionality to 2.8 defects). Among standard quality assurance techniques, only formal inspections have a higher effectiveness (60 percent). By comparison, test-driven development (TDD) comes in at 37 percent. These numbers are not additive, of course. Inspections plus static analysis don't deliver 100 percent defect-free code. By the same token, static analysis reveals problems that inspections and TDD cannot find. Let me explain.

Static-analysis tools today fall into roughly two tiers. There are the open source tools, which in Java are particularly good. These include FindBugs, PMD, walkmod,

and to a lesser extent [Checkstyle](#). Commercial tools—such as those from Coverity, Parasoft, Rogue Wave Software, and other vendors—are significantly more advanced. They can do magical things such as data-motion analysis, which reveals defects that are hard to see via other means. For example, this analysis can show that a given data item can never be null because all paths that touch it first pass through another module far afield that guarantees nonnullness. Or it can find the opposite, that a method that counts on being passed only prescreened objects can in fact be passed a null. Some of these tools excel at finding other very difficult bugs, such as unwanted interactions between two threads, or the rare case that a data item can be read incorrectly due to the design of the Java memory model. These are subtle items that can be hard to locate and costly to fix—and they're typically not revealed by formal inspections or unit testing.

The resistance to static analysis, I believe, stems from a reputation built up in its early days of delivering false positives—that is, claiming to find a defect where in fact none exists. Per Jones,

false positives a decade ago averaged 10 percent but now are well below 3 percent. In commercial products, the rate is even lower. The other perceived drawback is that the tools are slow. This remains true. You typically run the high-end tools once a day for the whole project, but always on code about to be checked in. The tools can analyze new code by cross-checking with the database of artifacts from the most recent whole-project scan.

Static analysis has an additional upside: It's not disruptive to existing site practices. You can add it to a testing pipeline with ease. This aspect and its remarkable effectiveness make it one of the simplest, but undeservedly underused, ways to improve quality.

Andrew Binstock, Editor in Chief
javamag_us@oracle.com
[@platypusguy](https://twitter.com/platypusguy)

PS: Update on our evolution: In this issue, you'll see that we updated our fonts to improve legibility and we've begun what will be a long-running series of articles on JVM languages. Let me know if you have any suggestions.

CREATE
THE FUTURE

oracle.com/java

20
YEARS
1995-2015

Java™

ORACLE®

Adobe ColdFusion Celebrates 20 Years of Making Complex Coding Tasks Easy

Originally developed by Jeremy Allaire and JJ Allaire in 1995 to make it easier for developers to connect simple HTML pages to a database, ColdFusion has been around almost as long as the web itself and predates many popular web development languages. A full scripting language—CFML—and an integrated development environment were added, and ColdFusion remains popular with coders for being



Tridib Roy Chowdhury, General Manager and Senior Director of Products, Adobe

a rapid web application development platform that helps them reduce development time by an order of magnitude.

A history of innovation

Over two decades ColdFusion has been at the forefront of simplifying all tasks programmers find tedious and time consuming, such as charts and graphs, PDF generation and manipulation, WebSockets and

many more. Many of the things that seem so obvious to us now were innovative solutions first offered by ColdFusion, allowing developers to create robust, scalable, high-performing, secure web- and mobile-based applications with minimal effort.

Each release has built on the previous ones, and ColdFusion has long since gone beyond its “tag-based scripting language” past. Today, ColdFusion can talk to pretty much everything, from legacy COM and CORBA to .NET assemblies and Java classes.

And it can be used to develop pretty much every application required to meet dynamic business needs—which is why it is so prevalent in large enterprises and government setups.

ColdFusion has a fan following in smaller organizations, too. “When we started out, as [with] any startup, the budget was always short. So we needed a language that allowed us to develop the application in a short period of time. ColdFusion dramatically cut down the development cost,” says Sumit Verma, co-owner at Ten24 Digital Solutions.

“ColdFusion has clearly stood the test of time,” says Rakshith Naresh, product manager at ColdFusion. “One of the reasons why customers keep coming back to ColdFusion is the productivity benefit it offers.”

Continuing to blaze the trail

“There really isn’t a comparison at all between how Adobe reacts to our needs versus any of the other software companies that we work with. Basically, what happens is ColdFusion allows us to change our market,” quotes Eric Kratz, president at VSR Systems.

Be it overhauling your company’s HR operations, updating your firm’s global intranet, or powering the world’s busiest electronic storefronts, you can be sure this 20-year-old will rise to the occasion.

“ColdFusion continues to do well while staying at the forefront of technology. The unique integration with HTML5 along with the end-to-end mobile development platform and the increased focus

on security in ColdFusion give enterprises the confidence and edge to easily adopt the latest technologies as they make their digital transitions. The future roadmap for improvement in the mobile development platform and the addition of social analytics while staying true to the ColdFusion credo of ‘making hard things easy’ indicate exciting times for ColdFusion in the coming years.”

Tridib Roy Chowdhury, General Manager and Senior Director of Products, Adobe

Adobe ColdFusion Summit 2015

(November 9–10, Las Vegas, Nevada)

brings together the web application community. Interact with ColdFusion experts, domain leaders, and peers, and learn about the latest technologies, techniques, and strategies to help you rapidly build and successfully deliver web applications to market. Explore how ColdFusion is driving change and how you can propel this momentum.

Adobe ColdFusion Summit is the largest summit for ColdFusion developers and has had three successful annual installments.

For more information, visit adobe.com/coldfusion



MAY/JUNE 2015

In the article “What’s New in JPA” in the May/June 2015 issue, I saw in Listing 17 that the author uses a cast to long. But in fact, if the code were

it would remove the need for the cast.

—Josh Toepfer

IoT Coverage

—Sam Desd

Editor Andrew Binstock responds:
“We had a long article on using Java
for IoT on the Raspberry Pi in the

Inside the CPU

—Yury Pitsichin

I was wondering if there is a possibility of getting a printed edition of the magazine, at least through a third-party provider? After a complete day of work in front of the computer, it's good to have a print magazine for a change of environment.

Publisher Jennifer Hamilton responds: “I totally understand wanting to disconnect and enjoy a printed publication. However, Java Magazine is being published only in digital format. When I want to read a hard-copy version of an article, I download the PDF and print the pages in landscape mode using the Fit option in Adobe Acrobat Reader.”

The magazine needs to be available in the Kindle format. I don't get why people would download a PDF, or read the same on their phones. It's too cumbersome.

—Jude Pereira

“In Silicon Valley, at least, most people in tech read magazines on tablets (which, in my opinion, are better for reading articles with code). For users of tablets, we offer both iOS and Android apps. For all others, we offer PDF. We are actively looking at expanding our list of distribution targets, and the Kindle is at the top of that list. However, we don’t expect to be able to provide that upgrade before the end of the year or early next year.”

We welcome comments, suggestions, grumbles, kudos, article proposals, and chocolate chip cookies. All but the last two might be edited for publication. If your note is private, please indicate this in your message. Please write to us at javamag_us@oracle.com. For other ways to reach us, please see the last page of this issue.

//events /

San Francisco, California



JavaOne 2015 *OCTOBER 25–29*

SAN FRANCISCO, CALIFORNIA

Join the single largest gathering of Java developers. From sessions, workshops, labs, and exhibits to keynotes and Birds-of-a-Feather sessions, learn about the latest language changes to improve coding efficiency. You'll also learn how to build modern enterprise and server-based applications, create rich and immersive client-side solutions, build next-generation apps targeting smart devices, and compose sophisticated Java web services and cloud solutions.

JavaDay Kharkiv

OCTOBER 1

KHARKIV, UKRAINE

Enjoy and learn in a full day of world-class talks. Topics include core JVM platform and Java SE (Java 8), JVM languages and new programming paradigms, web development, and Java enterprise technologies.

Silicon Valley Code Camp

OCTOBER 3–4

SAN JOSE, CALIFORNIA

Last year, 4,500 people

attended this free community event where developers learn from fellow developers. In addition to technical topics, speakers present on software branding and legal issues.

JAX London

OCTOBER 12–14

LONDON, ENGLAND

JAX London brings Java, JVM, and enterprise professionals together for a technology- and methodology-packed event. Participants get full access to Big Data Con London, which features modern datastores, big data architectures based on Hadoop, and advanced data processing techniques.

JDD

OCTOBER 13–14

KRAKOW, POLAND

JDD is a two-day conference for all Java enthusiasts, who can participate in more than 30 lectures, workshops, interactive trainings, and networking opportunities. JDD attracts speakers from all over the world and offers lectures in English.

GeeCON

OCTOBER 23–24

PRAGUE, CZECH REPUBLIC

Join more than 2,000 participants at GeeCON, which is focused on JVM-based technologies with special attention to dynamic languages such as Groovy and Ruby. GeeCON participants share experiences about development methodologies and craftsmanship, enterprise architectures, design patterns, distributed computing, and more.

W-JAX 15

NOVEMBER 2–6

MUNICH, GERMANY

The W-JAX conference covers current and future aspects of technologies such as Java, Scala, and Android. Also addressed are web applications techniques, agile development models, and DevOps.

J-Fall 2015

NOVEMBER 5

EDE, NETHERLANDS

The annual Java conference organized by the Dutch Java User Group (NLJUG) typically sells out and has outgrown its usual venue. This year, J-Fall

PHOTOGRAPH BY WESTEND61/GETTY IMAGES





will take place in the CineMec in Ede.

Devoxx Belgium

NOVEMBER 9–13

ANTWERP, BELGIUM

By developers for developers, this event has 200 speakers and 3,500 attendees from 40 countries. Tracks this year include Java SE, JVM languages, and server-side Java, as well as cloud and big data, mobile, and architecture and security, among others.

Devoxx Morocco

NOVEMBER 16–18

CASABLANCA, MOROCCO

Formerly the JMaghreb confer-

ence, this event is a university day of training, workshops, and labs followed by conference days of sessions on software development, web, mobile, gaming, security, methodology, Internet of Things, and cloud. The Decision Makers evening includes discussion of issues related to the IT industry in Morocco.

QCon San Francisco 2015

NOVEMBER 16–20

SAN FRANCISCO, CALIFORNIA

A practitioner-driven software development conference, QCon is designed for technical team leads, architects, engineering directors, and project managers who influence innovation in their teams. Tracks this year include Taking Java to the Next Level and The Dark Side of Security. The last two days are devoted to workshops.

Codemotion Milan

NOVEMBER 18–21

MILAN, ITALY

This conference is open to users of all languages and platforms. It offers full-day workshops on the first two days, followed by keynotes and conference sessions.

Codemotion Spain

NOVEMBER 27–28

MADRID, SPAIN

This two-day event draws nearly 2,000 attendees, represents more than 30 communities, and features coding lectures and workshops. Activities for startups, recruiting, and networking are included.

Clojure eXchange 2015

DECEMBER 3–4

LONDON, ENGLAND

Meet with the world's leading experts, learn how to use Clojure with your team, and discuss war stories with your peers. Both days will feature a mixture of talks covering various aspects of Clojure development: from libraries to music, from ClojureScript to data.

Groovy and Grails eXchange 2015

DECEMBER 14–15

LONDON, ENGLAND

Stay ahead of the curve and hear the 2016 roadmap for Groovy and Grails from core committers and Groovy authorities Guillaume Laforge and Graeme Rocher. Engage with other leading experts and fellow enthusiasts and learn the latest innovations and practices.

Apache Hadoop Innovation Summit

FEBRUARY 11–12

SAN DIEGO, CALIFORNIA

With presentations from more than 25 hands-on industry speakers, topics covered will include MapReduce and Spark, building privacy-protected data systems, scalable data curation, best practices, and architectural considerations for Hadoop applications.

Embedded World 2016

FEBRUARY 23–25

NUREMBERG, GERMANY

The 14th annual gathering of embedded system developers will explore the latest developments, define trends, and once again present the key areas of focus for future developments. This is where hardware, software, and system development engineers come together to turn the next milestones of the Internet of Things into reality.

Have an upcoming conference you'd like to add to our listing? Send us a link and a description of your event at least four months in advance at javamag_us@oracle.com. We'll include as many as space permits.

AOT Compilation Is Coming to Java 8

Excelsior JET 11 will support Java SE 8 and JavaFX 8 on all desktop platforms.

Get Your Early Access Copy Now

Registration-Free Download





10

My only reservation about the book is that it does not cover secondary Java EE technologies, such as Java Message Service or JavaMail. But these consider-

When we began doing critical book reviews in the May/June issue, we stated that due to possible conflicts of interest, we would not review books from Oracle Press. However, after securing approval for independent reviews from both Oracle and McGraw-Hill (the publishers of Oracle Press books), we begin in this issue to examine two recent titles from that imprint: one we liked a lot and one we did not. —Ed.

ations aside, this is an excellent programming introduction to Java EE 7. —Andrew Binstock



JAVAFX RICH CLIENT PROGRAMMING ON THE NETBEANS PLATFORM

Gail and Paul Anderson
Addison-Wesley

Of all the words in this book's title, *platform* most indicates its true content. This book is not a tutorial on JavaFX using NetBeans. Rather, it's a book about using JavaFX in conjunction with NetBeans as a software platform for developing desktop applications. In this sense, NetBeans as platform corresponds roughly to the Eclipse Rich Client Platform.

That is, it provides the software platform on which rich desktop apps can be built. NetBeans provides its own internal module

system (not Eclipse's OSGI) and exposes an extensive UI metaphor—which includes a windows manager with multiwindow support, search capabilities, a wizard builder, and so forth.

Under this presentation goodness is a lot of technology that delivers useful services: for example, an entire file-management layer, which includes goodies such as a file-status monitor that issues a message whenever a watched set of files is modified; RESTful web services; application update services; and so on.

The explanations of the best ways to exploit the NetBeans platform are clear, approachable, and illustrated with useful examples. The integration with JavaFX, however, feels somewhat rougher. Even though JavaFX is presented from scratch (presumably aimed at someone new to the technology), a beginner would be unlikely to be capable of following along without getting lost. The ideal reader is proficient with Java and has familiarity with writing UIs either in Swing or JavaFX. For such a reader, the JavaFX chapters will serve as a review and an update that enables safe passage to the discussion of NetBeans features. Fortunately, almost anyone

undertaking a project using the NetBeans platform will have that kind of background. For those readers, this book is a godsend.

The question I'm led to ask is, how many such readers could there be? I expect it's a small set, which makes me admire both the authors and the publisher for releasing a nearly 1,000-page volume on so narrow a topic. But for anyone in that group, this volume is *the* guide to have. —AB



INTRODUCING JAVAFX 8 PROGRAMMING

By Herbert Schildt
Oracle Press (McGraw-Hill)

As fun as JavaFX programming is, it is a world unto itself that requires a good guidebook to understand its ins and outs. But instead, this book is a most

incomplete introduction to the technology.

First, let me touch on what it doesn't cover, because these are crucial gaps: FXML and CSS styling. These two technologies, essential elements of any serious introduction, are dismissed via this glib line: "[All the examples in this book] are in Java. Therefore, no understanding of CSS or FXML is needed." The use of FXML and CSS precisely to avoid coding details in Java is apparently lost on the author.

Many other items are not covered. The author mentions JavaFX technologies that he tells readers to learn by themselves. However, if you turn to the section entitled "For Further Study" to do this, you find only a list of books all written by this same author *not one* of which is about JavaFX.

As to the actual content, the author is unhelpful. He covers the easy things in great detail, and the hard things are glossed over. There is precious little here that cannot be found in freely available tutorials. For serious developers, Hendrik Ebbers' *Mastering JavaFX 8 Controls* provides a far better introduction to JavaFX; it covers both CSS and FXML and does not skirt difficult material. —AB



EXCEL^{AT} ENTERPRISE & MOBILE DEVELOPMENT WITH A SMARTER IDE



IntelliJ IDEA

The most intelligent Java IDE

GET IT NOW

A free and open-source version is included www.jetbrains.com/idea

Testing:

WE'RE ALL IN THIS TOGETHER

Between the pre- and post-agile generations of developers, there is perhaps no greater difference than the role of developers in testing their code. The concept of developers writing tests as they pushed out code was a radical idea that took root and became an essential part of the coding process. Today some companies, such as Atlassian (page 42), are radicalizing this notion even further by moving *all* the responsibilities for QA to developers and training newly hired programmers from their first day in QA principles and techniques.

While agile values set the course in this new direction, it was undoubtedly the advent of JUnit—a fast test framework with intuitive mechanics—that made developer testing a universal reality. Despite JUnit's ubiquity, most of us, I fear, use only a familiar subset of its features and re-create capabilities already available. Our article on useful but underused features of JUnit (page 20) should help.

User interfaces are not as amenable to JUnit and so require specialized tools. For JavaFX, that tool is increasingly TestFX (page 14). For web-based interfaces, unfortunately, it is clusters of virtual machines exercising hundreds of combinations of browser releases and operating systems (page 26). Whatever your project's test tools, it is clear that we will not soon return to the days when coding and testing were segregated activities performed by different teams. I'm good with that. —Andrew Binstock

ART BY I-HUA CHEN

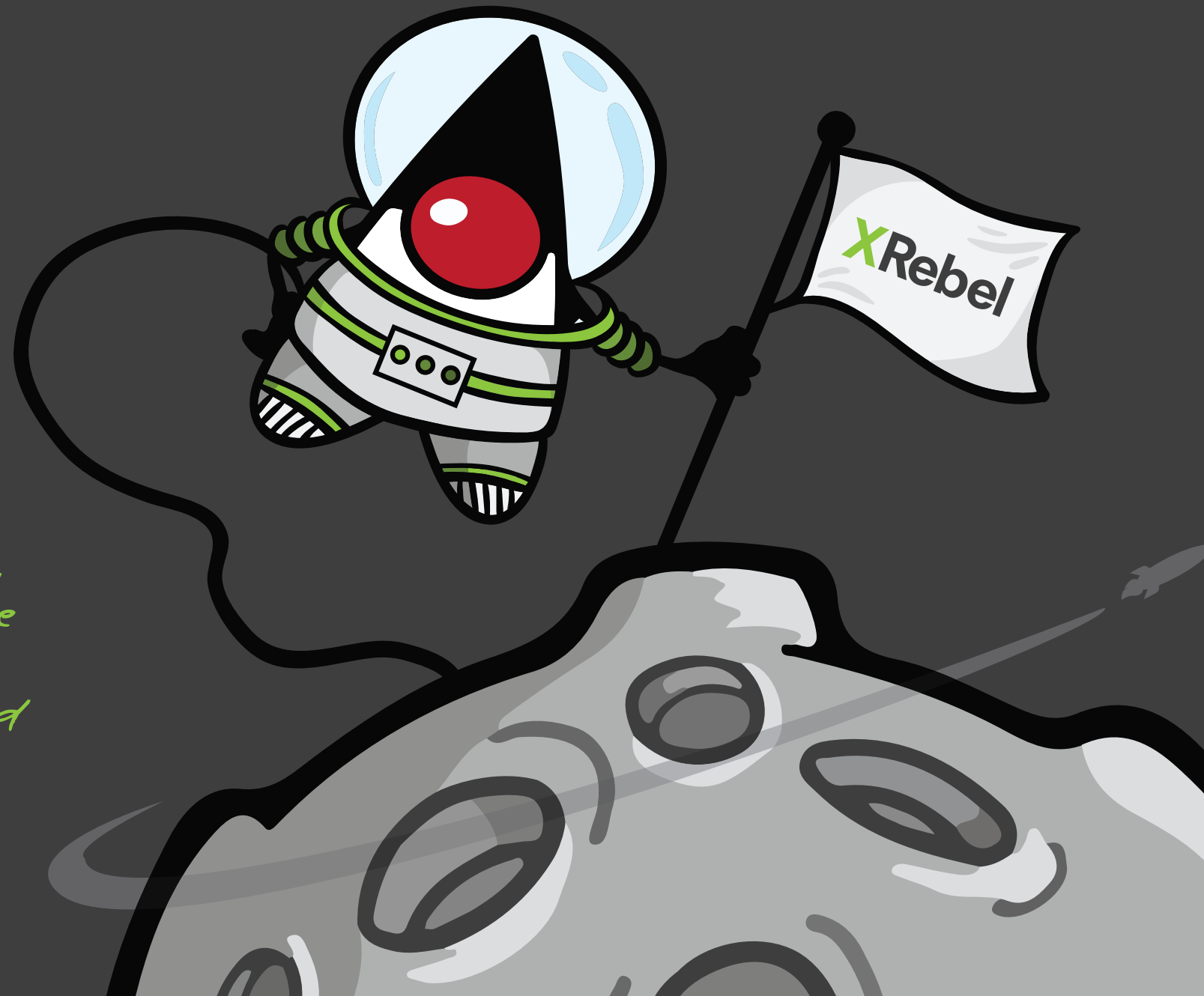


FIND ISSUES AS YOU CODE

with **XRebel** The Lightweight Java Profiler

TRY IT FREE NOW!

*One small step for Duke
one giant leap for Java-kind*





Make your testing a whole lot easier.

The `@Rule` annotation applies to a public field of a test

Since version 4.7, JUnit has offered the `TemporaryFolder` test rule, which provides a convenient way to create and manage a temporary directory. It guarantees that the new



NEIL MANVAR



Building and Automating a Functional Test Grid

How to assemble a grid for Selenium testing

On July 9, 2015, Apple released Mac OS X El Capitan beta, which offered significant new functionality. In this release, Apple introduced the mandate that all applications use SSL with a certificate that exceeds 1024-bit encryption. Google Chrome has also launched an update that now flags all SSL certificates that are from authorities not on the whitelist (that is, those that do not have public records) and certificates that use the (insecure) SHA-1 hash function.

What does all this have to do with functional testing? These seemingly innocuous changes will make users of Safari and the latest update of Chrome question security, and they could possibly raise some uncomfortable questions. The only way for IT organizations to spot these issues is by using a strong and current functional testing grid. In this article, I discuss how to set up your own testing grid for browser-based apps using [Selenium](#), the widely used open source testing tool.

The Grid

The idea of running manual tests on a browser of a particular type and version is straightforward. But applications in the wild do not run on just one type or version of a browser. No matter what browser your customers use, they expect a quality experience.

You soon find that the matrix of browser types and versions gets complicated quickly. Add the OS versions and configurations, Selenium versions, and individual browsers' web drivers, and you suddenly have an unmanageably complex test environment. This calls for automation. Without automation, a testing grid provides only modest help to an organization.

The Balance

When it comes to testing grids, the entire test suite should be run against the top 80 percent of browsers used. This rule is important, because testing on less common browsers has an effort cost that is greater than the impact of the potential issues that are found. The remaining browsers should be available for testing when specific issues arise, usually from some support request. The idea is to create a system where those browsers could be available for testing on demand.

According to the online course site W3Schools, its traffic shows that 97 percent of browser usage is shared among Chrome, Firefox, Internet Explorer, and Safari, in that order. Sites with a more business-oriented audience see a greater ratio of Internet Explorer, especially older versions of the browser. This shows that it is hard to predict what combinations of browser, version number, and operating system releases a given application must be tested for

The caveat to the 80-percent rule is effort cost. Most shops find that they don't even have the ability to test the top 50

A testing grid without automation
provides only
modest help to
an organization.

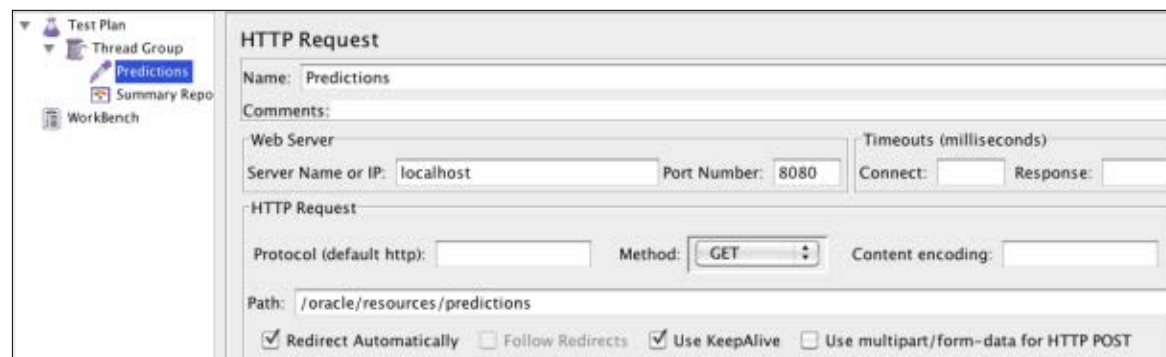


Figure 1. HTTP request configuration in JMeter

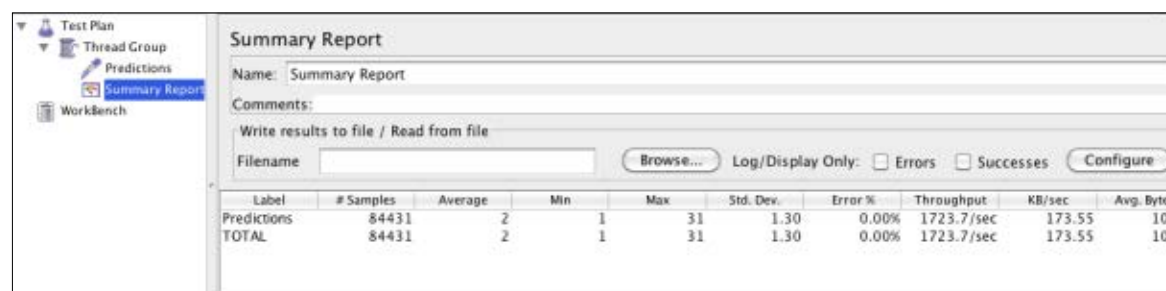


Figure 2. JMeter Summary Report



Figure 3. VisualVM CPU and memory monitoring overview

Sending a GET request to the URI `http://localhost:8080/oracle/resources/predictions` returns a **Prediction** entity serialized as:

```

{"prediction":
  {"result": "JAVA_IS_DEAD",
    "predictionDate":
      "1970-01-01T19:57:39+01:00",
    "success":"false"}
}.

```

Services provided by Java API for RESTful Web Services (JAX-RS) are easily stress-testable; you need only execute several HTTP requests concurrently.

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

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

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

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

VisualVM Turns Night to Day

GlassFish Server Open Source Edition 3.1.x and Java DB (the open source version of Apache Derby) are Java processes that can be easily monitored with VisualVM. Although VisualVM is

shipped with the current JDK, you should check the [VisualVM Website](#) for updates.

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

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

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

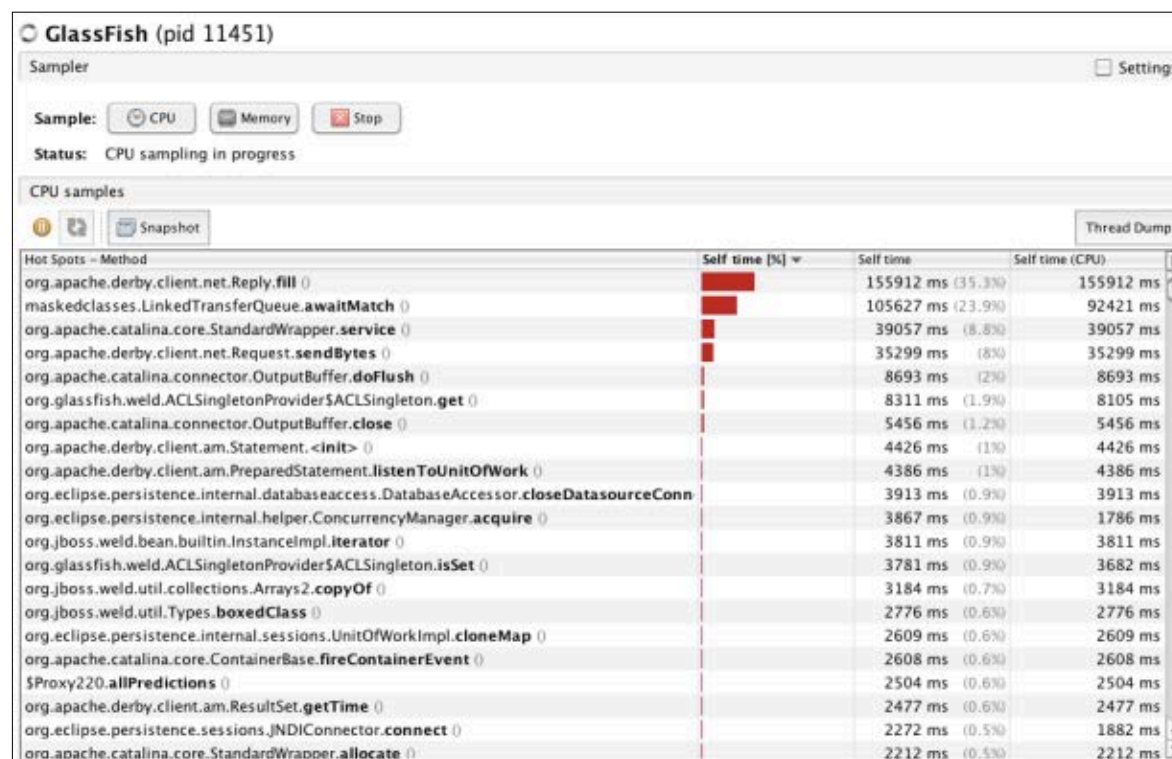


Figure 4. CPU monitoring with Sampler

load generators to the cloud.

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

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

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

The sampling overhead is about 20 percent, so with an active sampler, you can still achieve 1400 transactions per second. As expected, the application spends the largest amount of time communicating with the database.

How Expensive Is System.out.println?

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

■ Listing 1.

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


Figure 5. Performance degradation of 50 percent

Figure 6. CPU Sampler with System.out.println

Figure 7. Performance after reducing the pool size

Figure 8. Sampler output with two connections in the pool

```
WARNING: RAR5117 : Failed to obtain/create connection from connection pool [ SamplePool ]. Reason :
```



```
com.sun.appserv.connectors.internal.api.Pooling-
Exception: In-use connections equal max-pool-size
and expired max-wait-time. Cannot allocate more
connections.
```

```
WARNING: RAR5114 : Error allocating connection :
[Error in allocating a connection. Cause: In-use
connections equal max-pool-size and expired max-
wait-time. Cannot allocate more connections.]
```

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

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

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

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

■ Listing 3.

```
public Connection connect(Properties properties)
    throws DatabaseException, ValidationException {
    String user = properties.getProperty("user");
```


application server through different channels and APIs.

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

■ **Listing 6.**

```
public class DataProvider {
    public static final String BASE_URL =
        "http://localhost:4848" +
        "/monitoring/domain/server/";
    public static final String HEAP_SIZE =
        "jvm/memory/usedheapsize-count";
    private Client client;

    public Snapshot fetchData(){
        try {
            long usedHeapSize = usedHeapSize();
            //... other assignments omitted
            return new Snapshot(
                usedHeapSize, threadCount,
                totalErrors, currentThreadBusy,
                committedTX, rolledBackTX,
                queuedConnections);
        } catch (JSONException e) {
            throw new IllegalStateException(
                "Cannot fetch monitoring" +
                "data because of: " + e);
        }
    }

    long usedHeapSize() throws JSONException{
        final String uri = BASE_URL + HEAP_SIZE;
        return getLong(uri,"usedheapsize-count");
    }
}
```


40



Atlassian's developers are required to formulate tests before they code and after. QA is there to help—but not test.

In this article, I discuss the techniques we use to train developers, and how you can start thinking like a tester, too.

New developers joining our company are put through “boot camp”—a series of classes that get them up to speed as quickly as possible. Members of the QA team run a class entitled “Exploratory Testing Workshop for Developers.” The content is aimed at new hires, who might not have been expected to perform their own testing previously. More-experienced developers are always welcome to use the class as a refresher. The goal is to teach developers how to use manual testing effectively and deliver high-quality features.

Some developers have a preconception that exploratory testing means manual scripted testing, which is tedious, low-value, and as a rule should never be done by a human (because it can be automated).

The second half of the workshop invites attendees to give exploratory testing a try on a real feature, with QA guidance to avoid the mindless-clicking-around trap. Generally, we need to give hints like these:

- Don't test the "happy path." Instead, think about what *might* not work and what *should* not work (such as users with and without expected permissions, or conflicting concurrent actions on a multiuser system).



after all. This can be presented as a challenge: Assume that problems exist, and keep searching until you find them.

We measure the effectiveness of the team's quality process by two important metrics: the ratio of successful stories (those that did not result in a shipped bug) and the ratio of rejected stories (those where the DoT found an issue before shipping) to total stories.

Obviously, we want the first metric to be trending toward 100 percent. If we are shipping problems to production, then the DoT has not done a good job. QA's focus is on improving that with more workshops, more pairing, and so on.

Once the team is reaching the quality bar on nearly all stories, the second metric is of particular interest. The DoT adds value by rejecting stories that aren't up to par, but rejections are an inefficiency in the process. We want the rejection rate to trend toward zero. Ultimately, we want to remove the DoT step and have the original developer deliver high quality on the first go, not after someone else has pointed out problems.

Precoding Test Thinking

As the team matures, we focus more on prevention rather than detection. This means removing the DoT step—often against the wishes of the team, who come to see it as a safety net. We replace the postcoding DoT step by a precoding step we call a *QA kickoff*: the developers talk a QA engineer through the story, explaining what they plan to implement. Together the developer and QA engineer brainstorm test ideas, possible risks, and anything else related to the story.

Typically, the QA engineer asks lots of questions, such as, “What happens when you do X? What if we move Y to Z and then back again?” This discussion adds huge value. When

As the team matures, developers no longer **depend on the QA engineer being available for every discussion.**

developers are aware of edge cases and potential problems, they can code and test to mitigate them rather than reworking to fix them.

The outcome of the kickoff is a set of testing notes, which developers can use to determine when the story is done. Once they are confident, the story can be shipped. *Voilà!* We have removed the need for an additional testing step.

The format of kickoffs and testing notes varies according to the context, but here are some common elements:

Specific risks. What are the critical areas that must work? Which are the parts most likely *not* to work? What needs to be tested thoroughly?

Test ideas. What user scenarios should be covered? What should be automated? What level of automation is appropriate? What needs to be checked manually?

Notes. A kickoff often raises a lot of questions and details that do not form coherent test ideas but still are good to jot down at this point—for example, in-product analytics, feature discoverability and accessibility, and so on.

Again, we use metrics to validate that the kickoff process is achieving the quality goals of the team. As the team matures, developers can take on both roles in a QA kickoff—they no longer depend on the QA engineer being available for every discussion on every story.

The Big Picture

As a team succeeds with the processes described previously, the developers take on more of the testing activities, while the QA engineer takes on less. You might wonder what QA can do with all that free time. A high-performing team of developers means that QA can focus on big-picture quality improvements. Here are some examples:

Scale. We can increase the number of developers for whom a single QA engineer can provide value. At Atlassian, for example, the ratio is often between 1:10 and 1:20.

Knowledge. We can look ahead to see how the team can

meet customer needs better, and teach that to the whole team through the kickoffs.

Innovation. We can experiment with process changes and measure their impact.

Tools. We can identify ways to make testing easier for developers. For example, the Atlassian QA team built an Amazon EC2 service that provides quick and easy access to any supported browser for testing and troubleshooting.

Results

We've had great results with these techniques. They have enabled the QA function to scale effectively with a rapidly growing development team; individual QA engineers have successfully innovated and multiplied their value; and we have rendered obsolete much of the tedious busy work that is accepted as a necessary evil of the traditional QA role.

But perhaps most importantly, the team of (often skeptical) developers has enthusiastically accepted these changes. We've given them confidence that they can test their own work, deliver it to a measurable high standard, and make efficiency savings that increase the team's velocity without sacrificing quality. Internal surveys about the process and the team consistently show this.

If you are a developer who currently relies on a traditional testing phase to catch the bugs that you produce, you should question whether this is a satisfactory state of affairs. Which statement matches your mindset? “I’m a good developer; I don’t need to test” or “Developers are not good developers unless they can also test.” [</article>](#)

LEARN MORE

- [Further details on the Atlassian process](#) (by the same author)
- [“The Day the QA Died”](#) (another view of the same transition)

THE VIRTUAL JUG

The [Virtual JUG \(vJUG\)](#) is an online-only Java user group with a global audience in more than 100 countries. It is now 18 months old and already has more than 3,500 members. To date,



it has hosted 39 online sessions and runs at least two sessions every month, as well as a monthly podcast.

The most popular live sessions have included one by Java creator James Gosling on his new Wave Glider project, and a deep tech session on the Java memory

model with Oracle's Aleksey Shipilëv. The most watched session in the vJUG's short history is a look at 55 new features in Java 8 by Simon Ritter, who heads Java Technology Evangelism at Oracle. Sessions can be watched live or in replay at <http://virtualjug.com>. All of our speakers are interviewed after their sessions to talk about what makes them tick and to answer many probing and sometimes uncomfortable questions.

The vJUG has created a couple of new initiatives in 2015, including a new podcast called the Java Council. Its goal is to fill the gap left by the [Java Posse](#), which ended in 2014.

Another new initiative this year is the vJUG book club, which coordinates the worldwide reading of a chosen book, a review, and a discussion over several vJUG sessions. The subject of the book could be anything from a deep look at technology to time management. The first book will be *Effective Java, Second Edition* by Joshua Bloch. Which book will be next? The great thing is the community will decide! If you want to be part of a unique online JUG, [visit us](#).

erencing functions by name. Of course, a lambda expression can be passed in as well:

```
operate(2, 3, { x, y -> x +y })
```

These capabilities deliver an elegant way of doing function pipelining:

```
val numbers = 1..100
```

```
numbers.filter { it % 2 == 0 }
    .map { it + 5 }
    .forEach {
        println(it)
    }
```

One more issue we attack with Kotlin is null pointer exceptions. In Kotlin, by default, things cannot be null, meaning that potentially the only way we'd get a null reference exception would be if we explicitly force it.

```
var city = "London"
```

In the code above, `city` could never be assigned a null value. If we want it to be null, we need to go out of our way to be explicit:

```
var city : String? = null
```

where `?` indicates that a type can be nullable. When interop-erating with Java, we provide certain mechanisms to warn of possible null references, as well as providing some operators to make the code more concise, such as the *safe call operator*:

```
var file = File("...")
file?.length()
```

Because of the `?.`, this code would invoke `length()` on

`file` only if `file` were not null. The standard library, a small runtime that ships with Kotlin, also provides additional functions in this area, such as the `let` function, which when combined with the safe call operator allows for succinct code, such as the following:

```
obj?.let {
    ... // execute code here
}
```

This results in the code block executing if the object is not null.

One last thing worth mentioning about Kotlin is its ability to easily enable the creation of DSLs—without the overhead that necessarily comes with maintaining them or the language knowledge required to implement them. Top-level functions, higher-order functions, extension functions, and a few conventions, such as not having to use brackets when the last parameter to a function is another function—these features allow for creating rich DSLs that are strongly typed. The quintessential example is that of type-safe Groovy-style builders. The following function generates the expected HTML output:

```
html {
    head {
        title {+"XML encoding with Kotlin"}
    }
    body {
        h1 {+"XML encoding with Kotlin"}
    }
}
```

Kotlin has the ability to easily create DSLs—without the overhead that necessarily comes with maintaining them or the language knowledge required to implement them.

Growth and the Road Ahead

Over the past year, and despite not having released version 1.0, we've noticed a substantial growth and interest in Kotlin. There has been an increase in downloads and visits to the Kotlin site, as well as an increase in technical questions in both our forums and public venues such as StackOverflow.

We're close to reaching the first major release, and we've made significant steps toward that. Over the past few milestone releases, we've been removing and adjusting some things in the language to make sure that once we release,

we'll be fairly certain that what we ship is there to stay. As any language designer or developer knows, whatever goes in a language stays as baggage pretty much forever.

ily migrate code to newer syntax. We believe that this way, we create a smooth experience for developers that are already using Kotlin in production.

Beyond these quick fixes, we're also focusing on improving other aspects of tooling. For Kotlin to be successful, the entry barrier should be low in all aspects. That is why we not only provide tooling for IntelliJ IDEA, in both Ultimate and the open source Community Edition, but also for build tools such as Gradle, Ant, and Maven, as well as a simple command-line compiler. We've also released a preliminary version of Kotlin for Eclipse, and we're hoping that much like there are contributions to Kotlin in other areas, the community will contribute to Eclipse support as well.

In conclusion, we developed Kotlin for our own use primarily and are heavily invested in it. For us, it's a tool that we're using to drive our own business, which is developer tools. We already have several internal and public-facing web applications written in Kotlin. Some of our newer tools are being written in Kotlin and our existing tools, such as IntelliJ IDEA and YouTrack, are adopting Kotlin. </article>

[This article is the inaugural installment of a new series on JVM languages that will appear in *Java Magazine*. We will examine the full range of languages, from large commercial efforts to projects driven by determined groups of hackers. In the next issue, we'll cover Jython. —Ed.]

- [Kotlin home](#)
- [Kotlin on StackOverflow](#)
- [Kotlin on Reddit](#)

Let's create a method that looks for an element that starts with a given letter and prints it.

```
public static void pickName(
    final List<String> names,
    final String startingLetter) {
    String foundName = null;
    for(String name : names) {
        if(name.startsWith(startingLetter)) {
            foundName = name;
            break;
        }
    }
    System.out.print(
        String.format(
            "A name starting with %s: ",
            startingLetter));

    if(foundName != null) {
        System.out.println(foundName);
    } else {
        System.out.println("No name found");
    }
}
```

This method's smell can easily compete with passing garbage trucks. We first created a `foundName` variable and initialized it to `null`—that's the source of our first bad smell. This will force a `null` check, and if we forget to deal with it, the result could be a `NullPointerException` or an unpleasant response. We then used an external iterator to loop through the elements, but had to break out of the loop if we found an element—here are other sources of rancid smells: primitive obsession, imperative style, and mutability. Once out of the loop, we had to check the response and print the appropriate result. That's quite a bit of code for a simple task.

```
public static void pickName(
    final List<String> names,
    final String startingLetter) {
    final Optional<String> foundName =
        names.stream()
            .filter(name ->
                name.startsWith(startingLetter))
            .findFirst();
    System.out.println(
        String.format(
            "A name starting with %s: %s",
            startingLetter,
            foundName.orElse("No name found")));
}
```

Some powerful features in the JDK library came together to help achieve this conciseness. First we used the `filter()` method to grab all the elements matching the desired pattern. Then the `findFirst()` method of the `Stream` class helped pick the first value from that collection. This method returns a special `Optional` object, which is the state-appointed `null` deodorizer in Java.

The `Optional` class is useful whenever the result may be absent. It protects us from getting a `NullPointerException` by accident, and makes it quite explicit to the reader that “no result found” is a possible outcome. We can inquire whether an object is present by using the `isPresent()` method, and we can obtain the current value using its `get()` method. Alternatively, we could suggest a substitute value for the missing instance, using the method (with the most threatening name) `orElse()`, as in the previous code.

Collections are common in programming and, thanks to lambda expressions, using them is now easier and simpler in Java. **We can trade the long-winded old methods for elegant, concise code** to perform common operations on collections.

the JDK uses a `reduce()` method. Let's look at the more general form of the reduce operation.

As an example, let's read over the given collection of names and display the longest one. If there is more than one name with the same longest length, we'll display the first one we find. One way we could do that is to figure out the longest length, and then pick the first element of that length. But that would require going over the list twice, which is not efficient. This is where a `reduce()` method comes into play.

We can use the `reduce()` method to compare two elements against each other and pass along the result for further comparison with the remaining elements in the collection. Much like the other higher-order functions on collections we've seen so far, the `reduce()` method iterates over the collection. In addition, it carries forward the result of the computation that the lambda expression returns. An example will help clarify this:

```
final Optional<String> aLongName =
    friends.stream()
        .reduce((name1, name2) ->
            name1.length() >= name2.length() ?
                name1 : name2);
aLongName.ifPresent(name ->
    System.out.println(
        String.format("A longest name: %s", name)));
```

A longest name: Brian

As the `reduce()` method iterated through the collection, it called the lambda expression first, with the first two elements in the list. The result from the lambda expression is used for the subsequent call. In the second call, `name1` is bound to the result from the previous call to the lambda expression, and `name2` is bound to the third element in the collection. The calls to the lambda expression continue for the rest of the elements in the collection. The result from the final call is returned as the result of the `reduce()` method call.

The result of the `reduce()` method is an `Optional` because the list on which `reduce()` is called might be empty. In that case, there would be no longest name. If the list had only one element, `reduce()` would return that element and the lambda expression would not be invoked.

From the example, we can infer that the `reduce()` method's result is at most one element from the collection. If we want to set a default or a base value, we can pass that value as an extra parameter to an overloaded variation of the `reduce()` method. For example, if the shortest name we want to pick is "Steve," we can pass that to the `reduce()` method, like so:

```
final String steveOrLonger =
    friends.stream()
        .reduce("Steve", (name1, name2) ->
            name1.length() >= name2.length() ?
            name1 : name2);
```

Joining Elements

We’ve explored how to select elements, iterate, and transform collections. Yet in a trivial operation—concatenating a collection—we could lose all the gains we made from concise and elegant code if not for a

We invoked the `collect()` method on the transformed list and provided it a collector returned by the `joining()` method, which is a static method on a `Collectors` utility class. A collector acts as a sink object to receive elements passed by the `collect()` method and stores them in a desired format: `ArrayList`, `String`, and so on.

Here are the names, now in uppercase and comma-separated.

BRIAN, NATE, NEAL, RAJU, SARA, SCOTT

The `StringJoiner` gives a lot more control over the format of concatenation; we can specify a prefix, a suffix, and infix character sequences, if we desire.

Conclusion

As we've seen in this two-part series, lambda expressions and the newly added classes and methods make programming in Java so much easier and more fun, too.

Collections are common in programming and, thanks to lambda expressions, using them is now much easier and simpler in Java. We can trade the long-winded old methods for elegant, concise code to perform common operations on collections. Internal iterators make it convenient to traverse collections, transform collections without enduring mutability, and select elements from collections without much effort. Using these functions means less code to write. That can lead to more maintainable code, more code that does useful domain- or application-related logic, and less code to handle the basics of coding. [➤](#)

This article was adapted from [Functional Programming in Java: Harnessing the Power of Java 8 Lambda Expressions](#) with kind permission from the publisher, The Pragmatic Bookshelf.

LEARN MORE

- [Overview of functional programming](#)
- [Cay Horstmann's explanation of using lambdas in Java 8](#)





Contexts and Dependency Injection: The New Java EE Toolbox

This series of four articles attempts to demystify Contexts and Dependency Injection (CDI). In the previous [two articles](#), I discussed what strong typing really means in dependency injection and how to use CDI to integrate third-party frameworks. In this article, I focus on how to get loose coupling with interceptors, decorators, and events. The final article will cover the integration of CDI within Java EE.

Interceptors are a way to solve the problem of cross-cutting technical concerns by intercepting method invocation. *Decoration* is similar to interception but is applied to business concerns. CDI *events* bring about even more loose coupling by implementing the observer/observable pattern in a very easy way.

Let's start with an explanation of interception, which is used to interpose on method invocations. It is a programming paradigm that separates cross-cutting concerns from our business code. Most applications have common code that is repeated across components—the cross-cutting concerns. These could be technical concerns, such as logging the entry and exit from each method or logging the duration of a method invocation. Or they could be business concerns, such as to perform additional checks if a customer buys more than US\$10,000 of items or send a refill order when the inventory level is too low. Both technical concerns and business concerns rely on the container to do much of the legwork of implementation.

The container does the interception. We need to remember that CDI beans live in a managed environment known as a *container* or *bean manager*. This bean manager provides many services, one of which is the ability to intercept method invocation. As can be seen in **Figure 1**, Bean A and Bean B are both managed by the CDI container. When we invoke a method on Bean A or Bean B, we can ask the container to intercept the calls and process some business logic. This can happen before or after the bean method is invoked, so we can also add business logic when the invocation returns.

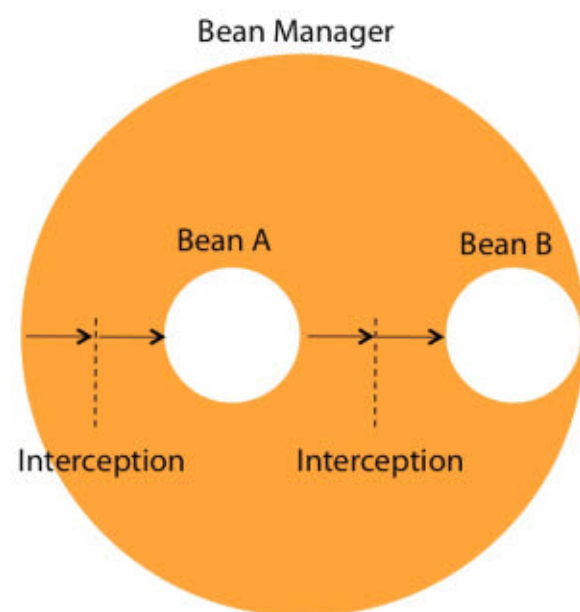


Figure 1. Intercepting method calls

An interceptor binding is just an annotation.

Take the `BookService` class defined in Listing 1. Let's say we want to log an entry each time the `createBook` method is invoked. A simple way of doing this is by injecting a logger and tracing the entry. It appears that this technical concern can also be used by other methods and other beans. Instead of copying and pasting code around, we can isolate this code in an interceptor, give it a name (here, it is `@Loggable`), and apply it on whatever method we want. This tells the container, "intercept this method, and do whatever you need to do to log an entry." This behavior can be extended to other technical concerns. In Listing 1, we tell the container that every method of the bean needs to be secured and transactional. `@Transactional`, `@Secured`, and `@Loggable` are called *interceptor binding*.

```
public class BookService {
```

```
        return new Book(
            title, price,
            generator.generateNumber());
    }
    @Loggable
    public Book raisePrice(Book book) {
        book.setPrice(book.getPrice() * 2.5F);
        return book;
    }
}
```

```
@InterceptorBinding
@Retention(RUNTIME)
@Target({METHOD, TYPE})
@Documented
public @interface Loggable {
}
```

Interceptor implementation. The `LoggingInterceptor` (see Listing 2) is a separate class, annotated with our `@Loggable` interceptor binding, but it also needs the special `@javax.interceptor.Interceptor` annotation. This will tell CDI that `LoggingInterceptor` is the interceptor called `@Loggable`. This class uses a logger to log method entries. Notice here that interceptors are CDI beans and can take advantage of dependency injection, for example. Despite being annotated with `@AroundInvoke`, the intercept method—which could be called whatever we want, by the way—must follow certain rules: The method must have an `InvocationContext` parameter and must return `Object`. But what is an *invocation context*?

more debug information and use another interceptor—let’s say `LoggingDebugInterceptor` (see [Listing 4](#), which I’ll explain shortly). How would we call its interceptor binding—`@LoggableWithDebug`? And if we have other logging interceptors, we would have as many interceptor bindings. Like qualifiers, we could create as many different interceptor bindings as we have implementations, or we could add members to our interceptor bindings, or we could aggregate them.

■ **Listing 4.**

```
@Loggable(debug = true)
@Interceptor
public class LoggingDebugInterceptor {

    @Inject
    private Logger logger;

    @AroundInvoke
    private Object intercept(InvocationContext ic)
        throws Exception {
        logger.info("> {}", ic.getMethod());
        logger.info("> Parameters          : {}",
            ic.getParameters());
        final Class<? extends Object> runtimeClass =
            ic.getTarget().getClass();
        logger.info("> Runtime class          : {}",
            runtimeClass.getName());
        logger.info("> Extended classes      : {}",
            new Object[]{runtimeClass.getClasses()});
        logger.info("> Implemented interfaces: {}",
            new Object[]{runtimeClass
                .getInterfaces()});
        logger.info("> Annotations ({})      : {}",
            runtimeClass.getAnnotations().length,
            runtimeClass.getAnnotations());
        final Class<?> declaringClass =
            ic.getMethod().getDeclaringClass();
        logger.info("> Declaring class      : {}",
            declaringClass);
    }
}
```

```

        logger.info("> Extended classes      : {}",
            new Object[]{declaringClass
                .getClasses()});
        logger.info("> Annotations ({}): {}",
            declaringClass.getAnnotations().length,
            declaringClass.getAnnotations());
        try {
            return ic.proceed();
        } finally {
            logger.info("< {}", ic.getMethod());
        }
    }
}

```

Interceptor binding with members. An interceptor binding is an annotation, so it can have as many members of any type as needed. Here, to differentiate between logging and debug logging, we could use a Boolean. Any other members of any data type are, of course, allowed.

```
@InterceptorBinding
@Retention(RUNTIME)
@Target({METHOD, TYPE})
@Documented
public @interface Loggable {
    boolean debug();
}
```

So now we have two interceptor implementations: `LoggingInterceptor` (see Listing 5) for normal logging and `LoggingDebugInterceptor` (see Listing 4) for extra debug logging. To differentiate them, we use our interceptor binding and set different values on the Boolean member: `debug = false` for `LoggingInterceptor` and `debug`

CDI events implement the observer/observable design pattern and are perfect for decoupling components that have no compile time dependency.

interceptors of the `createBook` method defined in Listing 8, `@Auditable` will be executed first, then `@Loggable`, and finally `@ThreadTrackable`.

■ Listing 10.

```
@Auditable
@Interceptor
@Priority(APPLICATION + 10)
public class AuditInterceptor {
    // ...
}

@Loggable
@Interceptor
@Priority(APPLICATION + 20)
public class LoggingInterceptor {
    // ...
}

@ThreadTrackable
@Interceptor
@Priority(APPLICATION + 30)
public class ThreadTrackerInterceptor {
    // ...
}
```

Decorators

Interceptors perform cross-cutting tasks and are perfect for solving technical concerns. By their nature, interceptors are unaware of the actual semantics of the actions they intercept, and therefore are not appropriate for separating business-related concerns. The reverse is true for decorators. Decorators are a common design pattern, initially described by the the [Gang of Four](#). The idea is to take a class and wrap another class around it. This way, when you call a decorated class you always pass through the surrounding decorator before you reach the target class, also known as the [@Delegate](#). Decorators are meant to facilitate adding additional logic to a business method. Interceptors and decorators, though similar in many ways, are complementary. Just

remember that interceptors are called before decorators when applied to the same method.

Figure 2 illustrates decoration. Here, the CDI bean `PurchaseOrderService` has one compute method that computes the total amount for a purchase order. An interceptor could intercept the call to this method and perform some cross-cutting technical logic. The decorator is slightly different because it is aware of the target's logic. For example, we could need a decorator that adds a discount to the purchase order for Christmas.

The diagram shows a large orange circle labeled 'Bean Manager'. Inside it is a white circle labeled 'Decorator'. Inside the 'Decorator' circle is a smaller white circle labeled 'Target'. The 'Target' circle contains the text 'compute'. The 'Decorator' circle contains the text 'compute' and 'PurchaseOrderService'. An arrow points from the 'compute' text in the 'Target' circle to the 'compute' text in the 'Decorator' circle. Another arrow points from the 'compute' text in the 'Decorator' circle to the 'compute' text in the 'Target' circle. A third arrow points from the 'compute' text in the 'Target' circle to the 'compute' text in the 'Decorator' circle. A fourth arrow points from the 'compute' text in the 'Decorator' circle to the 'compute' text in the 'Target' circle.

Figure 2. Inside a decorator

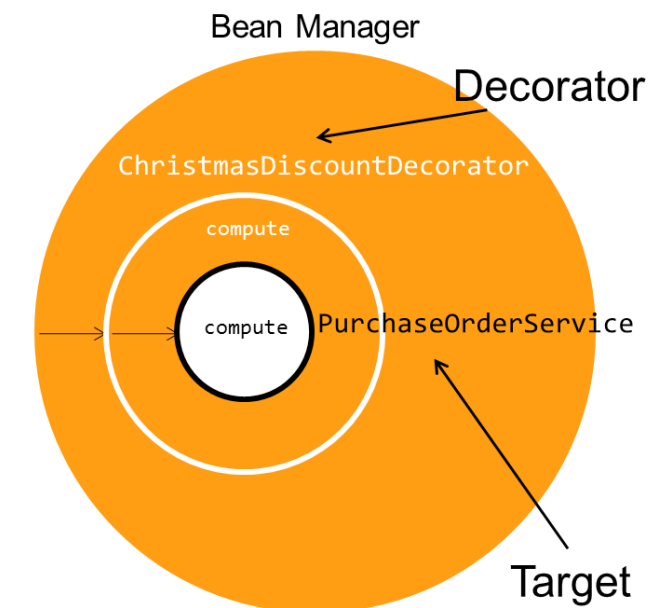


Figure 2. Inside a decorator

When we invoke the business method `compute`, a `ChristmasDiscountDecorator` intercepts the call, gets the total amount of the purchase order, and then applies a certain discount percentage. The `PurchaseOrderService` is called the target, and the `ChristmasDiscountDecorator` is the decorator.

Implementing the target. Now let's illustrate this business case with code. For decorators to be aware of the target's business logic, both the target and the decorator need to implement the same interface. In our example, we use the `Computable` interface (see Listing 11), which has a `compute` method that takes a list of items (let's say books), computes the price, and returns a purchase order.


```
@Inject
@Discount
private Float discountRate;

@Inject
@Delegate
private Comutable purchaseOrderService;

@Override
public PurchaseOrder compute(List<Item> items){
    PurchaseOrder po =
        purchaseOrderService.compute(items);

    po.setTotalAfterDiscount(po.getTotal() -
        po.getTotal() * discountRate);

    return po;
}
```

Enabling decorators. By default, all decorators are disabled like alternatives and interceptors, so we need to enable them using the beans.xml descriptor just by specifying the class name of the decorator implementation (see Listing 14). If we have multiple decorators, we can order them. And like interceptors, decorators can alternatively be enabled and ordered using the `@Priority` annotation.

■ Listing 14.

```
<beans xmlns="http://xmlns.jcp.org/xml/ns/javaee"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                           http://xmlns.jcp.org/xml/ns/javaee/beans_1_1.xsd"
       version="1.1" bean-discovery-mode="all">

    <decorators>
        <class>org.foo.bar.ChristmasDiscountDecorator</class>
    </decorators>
</beans>
```

Events

Dependency injection, alternatives, interceptors, and decorators enable loose coupling by allowing additional behavior to vary, either at deployment time or at run-time. Events go one step further, allowing beans to interact with no compile time dependency at all.

One bean can fire an event, and another bean can observe the event. The beans can be in separate packages and even in separate tiers of the application. This basic schema follows the observer/observable design pattern from the Gang of Four. Event notifications decouple event producers (the ones firing events) from event consumers (the ones consuming the events).

To illustrate event management, let's look at an example. In **Figure 3**, we have a **PurchaseOrderService** with a **create** method. When we invoke the **create** method, it creates a purchase order and then calls the **InventoryService** to update the item stock. As we've seen, CDI is the perfect framework to deal with dependencies. So here, **PurchaseOrderService** depends on **InventoryService**, and we could easily use **@Inject**. Then we realize that we need the **ShippingService** to ship items to the right location. Again, another **@Inject** could do. And what if we need to update some statistics of items sold, or invoke other services? Do we need to change the code of the **PurchaseOrderService** each time? Dependency injection enables loose coupling by allowing the implementation of the injected bean type to vary, either at deployment

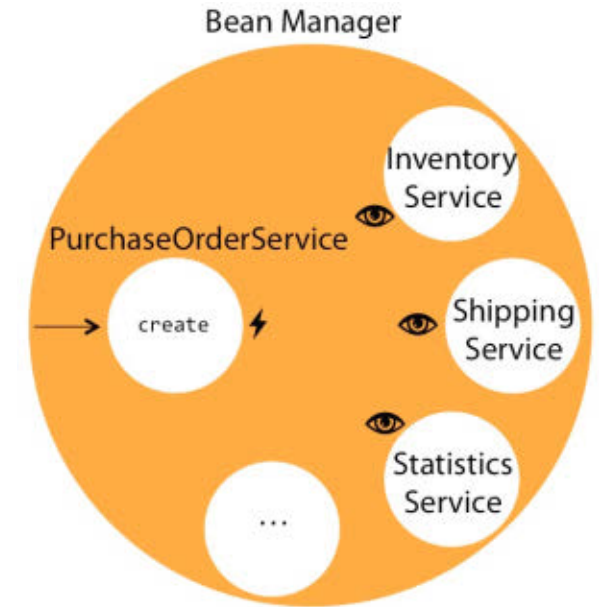
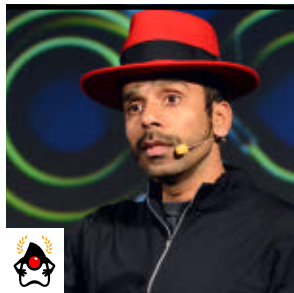


Figure 3. Loose coupling CDI-style

Event producer. Let's have a look at the event producer. The `PurchaseOrderService` is defined in Listing 15. The `create` method takes a list of items, does some business logic, and returns a `PurchaseOrder` object. Now, to fire an event is quite easy in terms of code. Event producers fire events using an instance of the `Event` interface. An instance of this interface is obtained by injection and is typed the same as the producer's object. In our example, we want to fire the `PurchaseOrder`, so `Event` is typed with `PurchaseOrder`.

68



ARUN GUPTA

BIO

A First Look at Microservices

The latest trend in enterprise computing is microservices. What exactly are they?

Customers expect a solution to be available on a variety of devices ranging from mobile clients and wearables to the Internet of Things. They expect a personalized experience and more-frequent updates. Businesses are trying to gain advantage over competitors with shortened time to market and a faster release cadence. Microservices are quickly emerging as part of the solution of this quest for speed of delivery.

Microservices Defined

Microservices are the product of an architectural approach that emphasizes the functional decomposition of applications into single-purpose, loosely coupled services managed by cross-functional teams for delivering and maintaining complex software systems quickly.

That's quite a lot of terms in this definition. Let's break them down and understand each one.

Architectural approach. Unlike Java EE, Spring, and .NET, the microservice architecture is a language-, platform-, and operating system-agnostic style that provides recommended guidelines on how applications need to be built.

Functional decomposition. A typical monolithic application in Java EE, for example, would consist of an EAR or WAR file. The entire functionality expected from the application is packaged in that one archive. There are several advantages of this approach, but the application tends to break down as it grows. Such an application needs to be decomposed into multiple WAR files where each WAR consists of a single function. This is where we get the concept of a microservice.

Single-purpose. Each decomposed part of an application is responsible for a single purpose only, and does it well. This smallness of scope is what the *micro* in *microservices* refers to.

Loosely coupled services. Service consumers do not have any knowledge about implementation of the producer service. Producers and consumers communicate only through a pre-defined contract. This makes the service loosely coupled,

which leads to more manageable, robust, and scalable applications. This also means that updating producer services doesn't require changing the consumer.

Decomposed applications are loosely coupled, which leads to more manageable, robust, and scalable applications. This means that updating one service doesn't require changing another service.

Cross-functional teams. Each microservice is created by a team that has a mix of different skills such as persistence, API design, and UI. This gives each team the competence to control the technology stack.

Complex software systems. The benefits of such an architecture style are evident only when building a nontrivial software system. This is so because

Each service can be independently deployed, and redeployed again, without affecting the overall system. This approach allows a service to be easily upgraded, for example, to add more features.



building a microservice simplifies the application code, but pushes the problem of plumbing to an integration layer and thus requires significant support from the infrastructure. Return on investment for a simple application might not be evident to begin with.

Characteristics

With this basic definition, let's examine the key characteristics of a microservices-based application.

- **Domain-driven design.** Functional decomposition of an application can be achieved using the well-defined principles of domain-driven design (DDD), by Eric Evans. Decomposition by domain is not the only way to break down the application but certainly a very common way. Each team is responsible for building the entire functionality around that domain or function of the business. Each team that builds a service includes the full range of developer skills, thus enabling a full-stack development methodology.
- **Single responsibility.** Each microservice should have responsibility for a single part of the functionality, and it should do that well.
- **Explicitly published interface.** Each service publishes an explicitly defined interface and honors that at all times. The *consuming service* cares only about that interface and should not have any runtime dependency on the *consumed service*. The services agree upon the domain models, API, payload, or some other contract; they communicate using only that. A newer version of the interface may be introduced, but either the previous versions continue to exist or the newer services are backward-compatible. Microservices may not

The scope of each microservice is much smaller than a monolith, and this leads to a smaller executable. **As a result, the deployment and the startup are much faster.**

break compatibility by changing contracts.

- **Independent DURS (Deploy, Upgrade, Replace, Scale).** Each service can be independently deployed, and redeployed again, without affecting the overall system. This approach allows a service to be easily upgraded, for example, to add more features. Each service can also scale independently using horizontal duplication or sharding. Implementation of the service, or even the underlying technology stack, can change as long as the exact same contract is published. This is possible because of the loose coupling.
- **Potential heterogeneity.** The implementation details of one service should not affect another service. This enables services to be decoupled from each other, and allows the team building a service to pick the language, persistence store, tools, and methodology that are most appropriate for them. A service that needs to store data in a relational database can choose MySQL, and another service that needs to store documents can choose MongoDB. Different teams can choose whatever technology is most efficient for them.
- **Lightweight communication.** Services communicate with each other using a lightweight communication, such as REST over HTTP. An alternative mechanism is to use a publish-subscribe mechanism that supports asynchronous messaging. Any of the messaging protocols, such as AMQP, STOMP, MQTT, or WebSocket, that meets the needs can be used here. Simple messaging implementations, such as ActiveMQ, that provide a reliable asynchronous fabric are quite appropriate for such usages. The choice of synchronous and asynchronous messaging is very specific to each service; even a combination of the two approaches can be used. Similarly, the choice of protocol is very specific to each service, but there is enough choice and independence for each team building a service.

Benefits

What are the benefits of a microservices-based application?

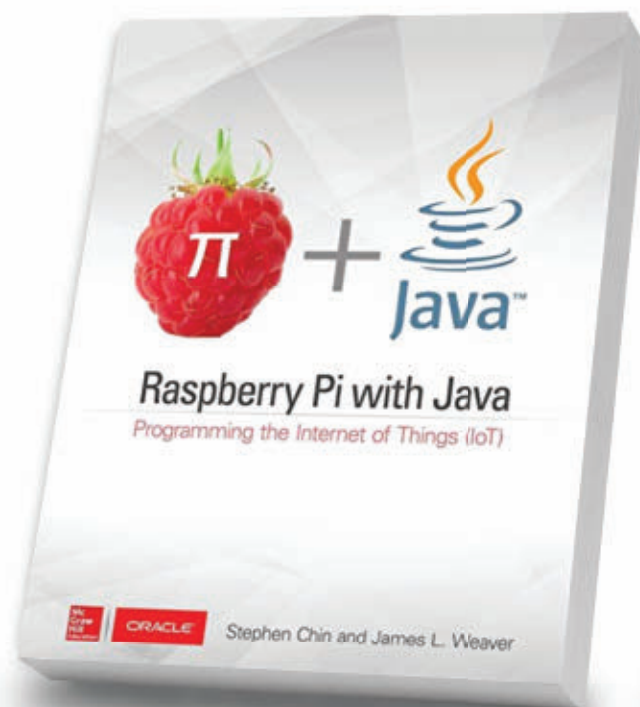
Ready or not, microservices are coming your way. Start thinking about them for your next project. Start with a pilot project, benefit from that first tryout, and then grow from there.

Microservices enable heterogeneity and polyglot stacks. This is possible because the only requirement is a well-defined interface, and the implementation is a detail left to each team

Happy microservicing! </article>

Your Destination for Java Expertise

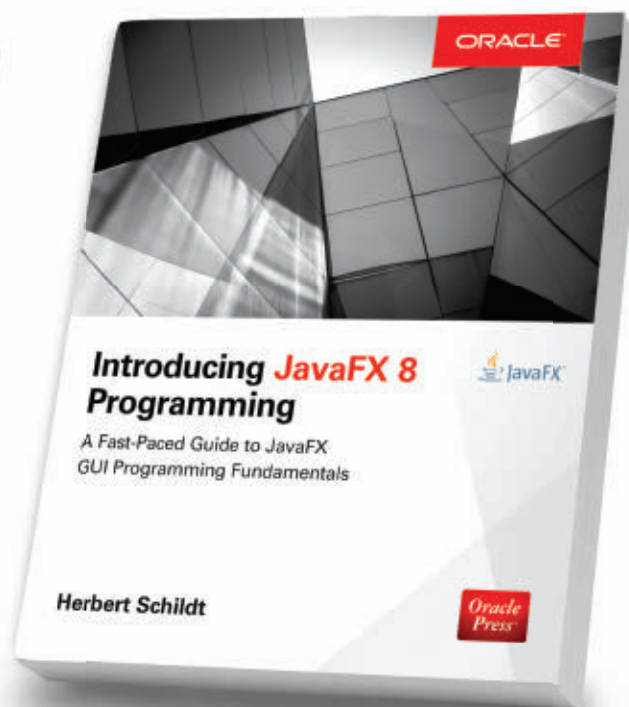
Written by leading Java experts, Oracle Press books offer the most definitive, complete, and up-to-date coverage of Java available.



Raspberry Pi with Java: Programming the Internet of Things (IoT)

Stephen Chin, James Weaver

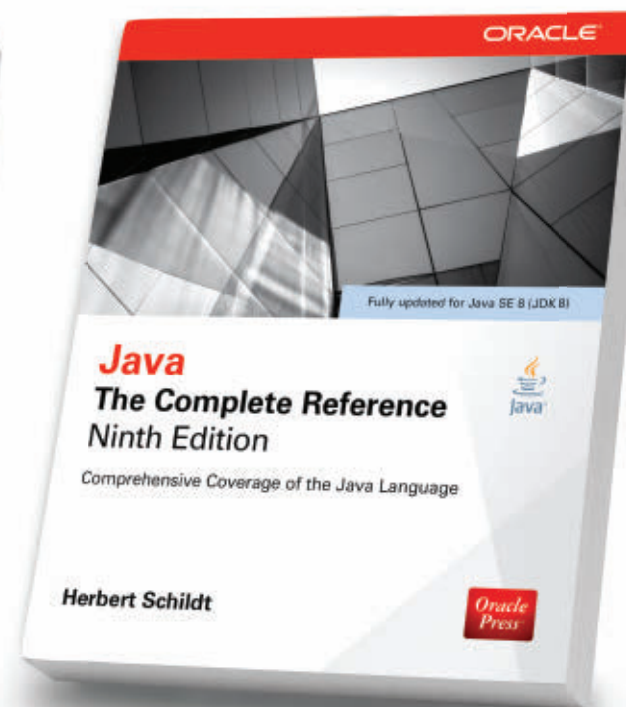
Use Raspberry Pi with Java to create innovative devices that power the internet of things.



Introducing JavaFX 8 Programming

Herbert Schildt

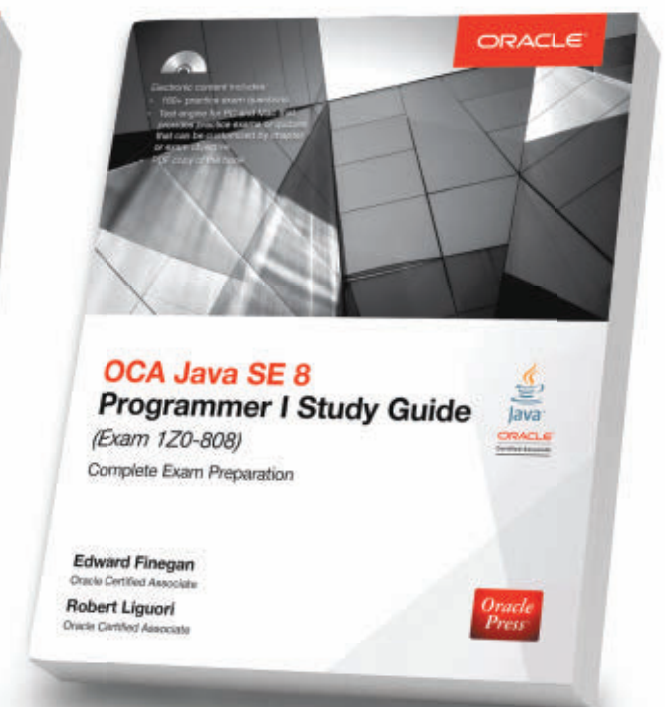
Learn how to develop dynamic JavaFX GUI applications quickly and easily.



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.



OCA Java SE 8 Programmer I Study Guide (Exam 1Z0-808)

Edward Finegan, Robert Liguori

Get complete coverage of all objectives for Exam 1Z0-808. Electronic practice exam questions are included.

Quiz Yourself

We all write code from within a subset of the full Java language. How well do we know that subset? Let's see.

The questions in this quiz are taken from certification test 1Z0-809: Oracle Certified Associate, Java SE 8 Programmer II, [Oracle Certified Java Programmer](#). The purpose of this certification is to enable beginners to demonstrate their knowledge of Java 8 concepts, somewhat above the fundamental level.

Naturally, these questions can have small unexpected traps that can snag even the attentive coder. But in this way, they reflect situations that occur in real life in which we wonder why code that looks right does not behave as expected. Ready? (Answers appear in the “Answers” section immediately after the questions.)

Question 1. Given this code fragment:

```
Map<Integer, Integer> codes = new TreeMap<>();
codes.put(1, 30);
codes.put(3, 20);
codes.put(2, 10);
codes.put(1, 40);
System.out.println(codes);
```

What is the result?

- a. $\{1=40, 2=10, 3=20\}$
b. $\{1=40, 1=30, 2=10, 3=20\}$
c. $\{2=10, 3=20, 1=40\}$
d. $\{1=40, 3=20, 2=10\}$

Question 2. Given this code fragment:

```
public abstract class Product { // line n1
    String name;
    Product(String name) {      // line n2
        this.name = name;
    }
    public final void printProduct() { // line n3
        System.out.println(name);
    }
    public void printLabel();      // line n4
}
```

Which line causes a compile-time error?

- a. line n1
- b. line n2
- c. line n3
- d. line n4

Question 3. Let's have a look at using the `Path` interface to operate on file and directory paths. Given this code fragment:

```
Path path = Paths.get("/home/user/./info.txt");
path.normalize();
System.out.println(path.getNameCount());
```

What is the result if the /home/users/./info.txt file does not exist?

- a. 3
b. 4

- d.** A `NoSuchFileException` is thrown at runtime.

Question 4. Now, let's filter a collection using lambda expressions.

Given this code fragment:

```
Stream<Integer> nS = Stream.of(5, 6, 8);  
// line n1
```

Which code fragment when inserted at **line n1** enables the code to print **1**?

- ```
a. List<Integer> oS =
 nS.filter(n -> n%2==1).toList();
 System.out.println(oS.size());

b. Stream<Integer> oS =
 nS.filter(n -> n%2==1);
 System.out.println(oS.count());

c. List<Integer> oS =
 nS.filter(n -> n%2==1).collect();
 System.out.println(oS.size());

d. Stream<Integer> oS =
 nS.allMatch(n -> n%2==1);
 System.out.println(oS.count());
```



**Question 1.** Option A is correct. The `TreeMap` instance is sorted according to the natural ordering of its keys. The keys are unique in all maps. The `put()` method replaces the previous value associated with the given key in the map and, therefore, 30 is replaced with 40.

Option B is incorrect. The keys are unique in a map object. 30 is replaced with 40. Option C is incorrect. The values in the `TreeMap`

instance are sorted according to the natural order of their keys. Option D is incorrect. The elements of the `codes` map are sorted in ascending order of its keys.

**Question 2.** Option D is correct: line n4 is invalid. A method that does not have its definition must be declared with `abstract`. To fix the compilation error at line n4, replace it with `public abstract void printLabel();`.

Option A is incorrect: line n1 is valid. A class declared with `abstract` can contain both abstract methods and concrete methods. Option B is incorrect: line n2 is valid. An abstract class can have a constructor. Option C is incorrect: line n3 is valid. An abstract class can have final methods.

**Question 3.** Option B is correct. The program prints the number of elements in the path string.

Options A and C are incorrect. On the second line, the path is normalized. The `Path` instance is immutable. Therefore, the `path.getNameCount()` method returns the number of elements in the path string. The program does *not* print the count of path elements to be redundant. Option D is incorrect. The `Path` instance is the string representation of the given path. The `/home/users/.info.txt` file need not be available in the file system. Only when `path.toRealPath()` is used and the file is not available in the file system is a `NoSuchFileException` thrown.

**Question 4.** Option B is correct. The `filter()` method returns a stream. Option A is incorrect. The `filter()` method returns a stream. A stream can be converted to a `List` type using the `collect()` method with the `Collectors.toList()` parameter. The `toList()` method cannot be used to convert a stream into a list. Option C is incorrect. The `collect()` method performs a mutable reduction operation on the elements in this stream using a `Collector`. It requires the `Collectors.toList()` parameter to convert the stream into a `List` instance. Option D is incorrect. The `allMatch()` method returns a `boolean` based on the provided predicate test.



# More Ideas to Boost Your Developer Career

Think about the best developers you know—people you admire for the work they do. Maybe you follow them on Twitter, watch their presentations, or read some of their articles or blog posts. Have some of them made an impact on your life and career? Why—because they write amazing code? Chances are that this is not the only thing they do to inspire you to follow and admire their work. The truth is that a flourishing developer career involves much more than just writing great code.

This series of articles focuses on three areas that we think have equal weight in a developer's career: code, community, and an open mind. Focusing on one while ignoring the others is a bad practice that limits personal and professional growth.

The good news is that changing your mindset does not require a huge effort. It is something you can do no matter what project you are working on—or what your employer thinks about it. It doesn't take a lot of time, and it is really a lot simpler than most things you deal with in your daily job. The best way to steer something in the right direction is to get it moving first. By taking small daily steps in the right areas, you can continuously push your career forward and achieve your goals.

We've already seen that to code better every day you need to practice: code, code, and code. And what's the purpose of

code? To solve problems and create solutions that can benefit people. Many developers don't even think about it, but we have a development environment available literally at our fingertips: the command-line interface (CLI). The CLI gives you the ability to control your whole environment. Menus and icons of tools, the IDEs, and the operating systems give you easy access only to what is visible. The CLI gives you access to everything.

Do you think of the CLI as a development environment? We do. You have to deal with the syntax and know the statements to execute. There is data and state, and you can process information. You even get unexpected results, and interpreter errors. That sounds exactly like coding!

The CLI provides a way to code and create automated software solutions. We can use it to create even more code: it boosts our productivity through automation. After all, this is how software enables our lives.

To be effective, the CLI requires you to learn, experiment, and remember what is available and how to use it. Get proficient with it. Experiment every day with new things you can do on the command line. And try to automate actions you need to perform repeatedly.

**ACTION ITEM:** Explore how many steps of your daily routine you can do through the CLI.

**ACTION ITEM:** Join some of those steps into scripts that automate parts of your work.



**BONUS ITEM:** Find a tool you already use that has a good CLI (such as Jenkins and AWS). Experiment to see how far can you go in using the CLI.

We all learn from content that other developers created—videos, webinars, blogs, talks. Usually these are the developers we look up to. The ability to communicate clearly is a terrific asset for your career, as is the wisdom to focus on and deliver pertinent and interesting information.

Even with these benefits, presenting a talk is still intimidating. To conquer your fear, remind yourself that presenting is a big challenge for most people. Even seasoned speakers get butterflies before walking on stage. Don't get discouraged.

A neat trick to remember: When people attend your talk, it is because they want to learn about your topic. You probably know more about it than they do; otherwise, they would be doing something else. Remember that the value you bring to your audience is your experience.

**BONUS ITEM:** Experiment with getting this talk approved. Submit it to a conference or tradeshow, even if the event is one that you don't expect to accept it—maybe an event in another country. Add your blog post or your recorded presentation as supporting material.

DevOps is the idea that Developers and Operations (every-

one, in fact) work together to deliver software. DevOps gets us to think differently. It makes delivering value to the users our top priority. When the whole company works together toward the goal of people using the software, we focus more on the benefits to our users and less on the fears that prevent us from doing it. We become better developers.

DevOps is an all-encompassing skill. It requires us to improve everywhere—from our tools and collaboration to our code and tests. And it is easy to start. Think big, but start small. Change your attitude. Focus on automation and delivery. Others will follow.

**ACTION ITEM:** Automate something that you do manually, such as your build, your deployment, or one of your high-level tests and test suites.

**ACTION ITEM:** Define something useful to measure, such as the number of bugs in the project, or how long it takes to deploy a new version. Start tracking this for the next few weeks. Add new metrics later.

**BONUS ITEM:** Learn one automation tool that you currently don't use, such as Jenkins, Ansible, Chef, or Selenium. Go beyond the tutorial: Run a small but real test to automate one thing in your project.

## Pulling It All Together

The skills described previously are easily combined. Using the CLI to automate tasks is a useful DevOps skill. Spending time automating your deployment will not only get you moving in the right direction, but it will also make your life easier. Consider delivering a brief presentation to your coworkers demonstrating what you did. You can then pass along this important skill by presenting the same talk to students in a nearby university.

It isn't difficult to steer your career in the right direction. Keep your mind open to new possibilities, and experiment with new ideas. Engage with the community in ways that are positive for you and others. And keep coding! </article>

