DZone

THE 2018 DZONE GUIDE TO

Java

FEATURES, IMPROVEMENTS, & UPDATES

VOLUME IV

Dear Reader,

In the mid-nineties, James Gosling and friends presented Java to the world. I remember downloading one of the first JDKs using my 56k dial-up connection, which took me a few attempts and left my parents unamused over the phone bill.

Still, I had loads of fun with it. The documentation, in general, was impressive at that time, and the already-famous *Java Tutorial* was fantastic and taught me about object-oriented programming and a few other very interesting subjects. There were also several code samples ranging from sorting algorithm implementations to a fractal generator. A disclaimer stating you couldn't use such examples for aircraft control and navigation, or to design and operate nuclear facilities, accompanied each example, which I thought was pretty funny. Who in their right mind would want to control an aircraft using a tic-tac-toe game?

After I somehow decided not to care about computer science anymore, I completely ignored Java for a couple of years — until, at the turn of the century, a stroke of sanity made me apply for college and pursue a software engineering degree. I re-encountered Java in my life, and this time it wasn't alone. There were a vast community and an entire ecosystem of open-source projects such as application containers, ORMs, IOC frameworks, 3D libraries, and general utilities, many of which are still around today, which is already impressive by itself. However, here's what surprised me the most: I picked up coding in Java almost as if I had never stopped in the first place.

Java is a simple language in nature with excellent type safety, excellent tooling, and, again, a fantastic community and a splendid ecosystem of libraries and utilities. Another often-overlooked characteristic of the platform is that both Sun and now Oracle always strived for 100% backward-compatibility, although sometimes at the expense of having extra verbosity, looking outdated, or having to maintain poorly designed APIs and features. With this in mind, it's easy to understand how it became so popular and how the JVM is the platform of choice in the enterprise and how it's what many startups end up picking up after the first few years going with the trends.

Welcome to the fourth volume of the *DZone Guide to Java*. You already know the history, so we want to show you the future, what's shaping up in the next releases, and what's cooking in the community. The Java platform is evolving rapidly, is learning from the past, and is embracing the future, and if you work on top of the JVM, you can rest assured you'll always have your spot in the front row at the technology theater.

Enjoy the reading.

By Hernani Cerqueira
DISTINGUISHED ENGINEER AT DZONE INC.

Table of Contents

Executive Summary

BY JORDAN BAKER, CONTENT COORDINATOR, DZONE

Java has long been one of the most, if not the most, ubiquitous and powerful programming languages. While the competition for developers' hearts and minds is increasing with the proliferation of languages like Golang, Java remains the most used language in the industry by far. For this year's Java Guide, we asked 935 developers for their thoughts on the state of the language, how they and their organizations use Java in development, what JVM languages they prefer, and more.

JAVA CONTINUES TO BECOME MORE FUNCTIONAL

Data: Of the new features included in Java 10, 47% of survey respondents told us they are using lambdas, 45% are using streams, and 34% are using optionals. All three of these features constitute aspects of functional programming. Indeed, 57% of respondents told us that they feel that they write more functional code in Java 10 than in previous versions.

Implications: This increase in the ability of developers to use Java for functional programming constitutes the continuation of a trend we've seen over the last several years. Last year, 51% of the survey respondents told us they were either comfortable or very comfortable using functional programming; this year, 52% expressed this same sentiment. And on top of that, 44% of this year's respondents told us that Java 10 has made coding in Java more fun.

Recommendations: 83% of respondents are using Java 8 or later to build new applications. This means that we can expect to see the rise of functional programming, and developers' comfort in mixing OOP and FP principals, continue. As such, if you have not already had a chance to hack away with Java 10, checking out its improved lambdas, streams, and optionals is something to consider.

JAVA EE VS. SPRING

Data: The rift we've historically seen between Java EE and Spring adoption in enterprise application development continued this year. Spring 4.x adoption fell slightly, from 47% in 2017 to 43% in 2018, and Spring 5.x garnered 35% of respondents' votes. While the use of Java EE has continued to grow, the release of Java EE 8 in late 2017 caused a split in this community, with 37% still opting for version 7, while 36% have made the switch to Java EE 8.

Implications: While 44% of respondents said the "new style" Java 10 has made coding more fun, 49% expressed no opinion. This could be reflected in the varied adoption in the latest version of Java EE. While the latest versions of Java and Java EE continue the trend toward greater functional programming, they don't appear to be as big of a paradigm shift as Java 8 and Java EE 7 were at the time of their release.

Recommendations: Though both Spring and Java EE have released new versions within the last year, Spring 4.x and Java EE 7 still hold considerable sway. As noted above, however, Spring 4.x usage continues to decline, down 4% from 2017 and 6% from 2016. Thus, it seems reasonable to expect to see Spring 5.x continue to grow in popularity — and considering a move to Spring's latest iteration is worth looking into. If you fall on the Java EE side of the divide, Java EE 8 has taken a big chunk out of Java EE 7 usage rates, with Java EE 7 dropping from 51% adoption in 2017 to 37% in 2018. While just 36% of the community currently uses Java EE 8, it seems to be the way of the future.

DEV AND PROD ARE GROOVY (FOR NOW)

Data: It seems that every year, the main JVM languages remain the same; and this year is no exception. When it comes to coding in development and production environments, Groovy reigned supreme, with 21% of respondents using Groovy in dev (though this down from 23% in 2017), and 15% using Groovy in production (though, again, this is down from 16% in 2017). Interestingly, when it comes to personal projects, Scala (29%) and Kotlin (20%) both faired far better than Groovy (15%).

Implications: While Groovy is still the most heavily used JVM language, Kotlin is quickly making up ground. In 2017, 4% reported using Kotlin in development, and 2% in production. This year, 13% are using Kotlin in dev and 6% in production. While Kotlin is quickly growing, both Groovy and Scala are stagnating. In 2017, 23% used Groovy for dev and 16% for production; in 2018, 21% use Groovy for development and 15% for production. Scala's use in production environments remained at 9% year-over-year, while its use in dev grew slightly from 15% to 16%.

Recommendations: While Groovy remains the most used JVM language in production and development environments, Kotlin is making a go of it. In last year's Java Guide, we noted Kotlin's marked growth, and that trend has continued on into this year. With Kotlin seeing such dynamic growth several years in a row, and the stagnation of its top competitors, getting your development and production teams training on Kotlin and beginning to work it into your codebase as your main JVM language is a move that could yield fruit over the next several years.
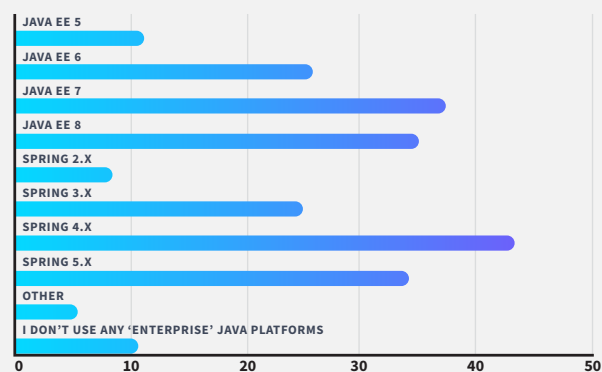
Key Research Findings

BY G. RYAN SPAIN

PRODUCTION COORDINATOR, DZONE

507 software professionals completed DZone's 2018 Java survey. Respondent demographics are as follows:

- 41% of respondents identify as developers or engineers; 20% identify as developer team leads; and 17% identify as architects.

- The average respondent has 13.7 years of experience as an IT professional. 58% of respondents have 10 years of experience or more; 21% have 20 years or more.

- 41% of respondents work at companies headquartered in Europe; 31% work at companies with HQs in North America.

- 21% of respondents work at organizations with more than 10,000 employees; 18% at organizations between 1,000 and 10,000 employees; and 23% at organizations between 100 and 999 employees.

- 87% develop web applications or services; 53% develop enterprise business apps; and 22% develop native mobile applications.

JAVA -VERSION

Since DZone's 2017 Guide to Java Development and Evolution was released, Java has continued to… evolve. Between the publication of that guide and the guide you're reading now, Java 9 and Java 10 have been released, and public updates for Java 9 have ended. By the time DZone's next Java Guide is released, it is likely that Java SE 11 will have officially been released (about 3 months from this writing); public updates to Java 10 will have ended (also about 3 months from this writing); and public updates of commercial-use Java 8 will have ended (about 6 months from this writing).

The change in Java's release cadence likely has a direct impact on the responses we saw in this year's survey regarding Java release adoption. Survey responses were collected roughly one month after Java 10 was released, and about 7 months after Java 9's release. 30% of respondents this year said they are using Java 9 or above, and these respondents are almost exclusively using Java 9+ in new (rather than existing) apps–only

GRAPH 01. Which of the following 'enterprise' Java platforms do you or your organization use?
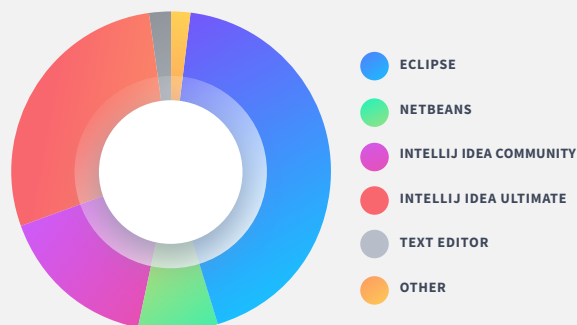

GRAPH 02. JVM language usage
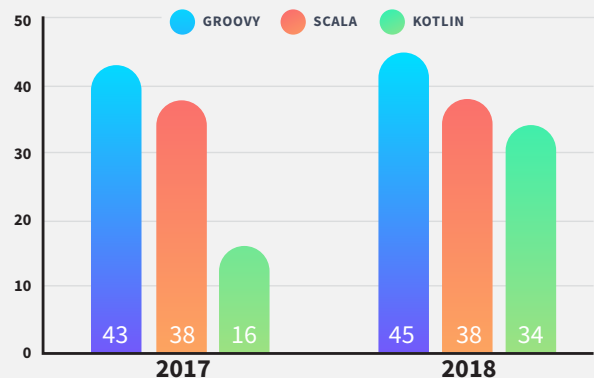
6% of respondents say they are using Java 9 or above in existing applications. Java 8, meanwhile, remains the predominant version of Java, with 92% of respondents saying they use Java 8 in some way. While respondents who said they use Java 8 in new apps fell from last year's results (89% in 2017 to 77% in 2018) as adoption of new Java versions takes off, there was an increase in respondents who said they use Java 8 for existing apps, from 49% to 61%. This is on par with the number of respondents who said they were using Java 8 for new apps in DZone's first Java survey, which was released about a year and a half after Java 8's release in 2014. Only 11% of respondents said they are using Java 7 or below for new apps, down from 19% last year.

KOTLIN, KOTLIN, KOTLIN!!!

Overall, JVM language adoption has not seen a lot of growth over the last few years. The two (non-Java) JVM giants–Groovy and Scala–saw fluctuations in survey responses from 45% (2016) to 43% (2017) to 45% (2018) for Groovy, and 41% (2016) to 38% (2017 & 2018) for Scala, these shifts being well within the survey's margin of error, showing no significant change in adoption of these technologies. Kotlin, on the other hand, has seen extraordinary adoption since 2016. 2016's Java survey saw 7% of respondents using the JVM-based language; this grew to 16% in 2017 and now to 34% in 2018. This means Kotlin adoption among our respondents has more than doubled each year for the past two years. Of course, it's impossible to directly correlate adoption rates of these technologies, considering Groovy and Scala were first released in 2004, giving plenty of time for extra hype to fade, while Kotlin was first released in 2011 and open-sourced in 2012. But Kotlin has certainly surpassed other JVM-based languages like Ceylon and Clojure to be among the top JVM languages out there; and with Kotlin's appeal for Android development, it's likely that its popularity will continue to grow.

THE ART OF JAVA IS LARGELY THE ART OF PERSISTENCE

Hibernate JPA remains the most popular persistence tool, with 50% of respondents using Hibernate's implementation of the Java Persistence API (up a negligible 1% from last year's survey). Standard JPA, on the other hand, fell from the second-place position it held last year, with respondents who said they use the tool dropping from 43% in 2017 to 35% in 2018, causing standard JDBC (42% in 2017 and 45% in 2018) to take it's spot as runner-up. A steep increase in adoption of Spring's JdbcTemplate from 26% in 2017 to 37% in 2018 pushed it to third place. This year's survey also saw an increase in respondents who said they use Spring Data, from 24% to 32%. While still not as popular as the other four persistence tools mentioned, this increase in Spring Data adoption hints at a rise in Java applications including more non-relational storage models.

FRONT-END FREE-FOR-ALL

This year's Java survey saw several shifts in responses regarding tools used for creating application front-ends. Respondents who said they use JavaFX fell to 11% from last year's 17%, putting JavaFX slightly below Swing at 14%. Respondents using the JavaServer Faces framework decreased dramatically, from 31% in 2017 to 21% in 2018, and the Struts MVC framework saw a slight decline, from 14% to 10%; however, Spring's MVC framework adoption increased from 33% to 39% this year. The use of JavaScript frameworks to handle Java app front-ends continued to rise this year, with React seeing a huge boost, jumping from 19% last year to 31% this year. Angular usage also increased from 52% to 57%. As development of web applications over desktop apps grows more and more common, this trend is likely to continue.

GRAPH 03. Where do you primarily write Java code?

- ECLIPSE
- NETBEANS
- INTELLIJ IDEA COMMUNITY
- INTELLIJ IDEA ULTIMATE
- TEXT EDITOR
- OTHER

GRAPH 04. Java data persistence tool usage

- HIBERNATE JPA
- STANDARD JDBC
- SPRING JDBC TEMPLATE
- STANDARD JPA
- SPRING DATA [ANY]

2017: 49, 42, 26, 43, 24
2018: 50, 45, 37, 35, 32
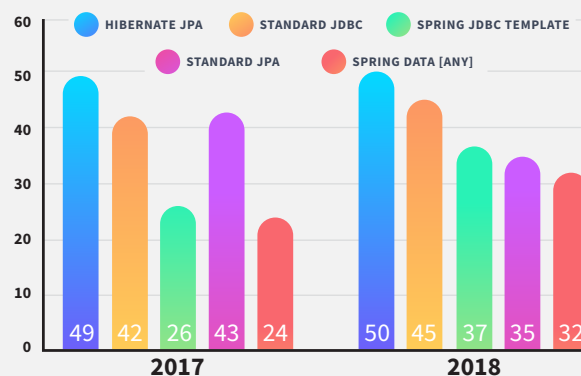
Null-Safe Programming: The Kotlin Way

QUICK VIEW

01. Understand the problem of `null` references and how they cause bugs in Java.

02. Learn how `Optional` can help avoiding certain `NullPointerExceptions` by making nullability explicit.

03. Take a look at Kotlin's type system and how it handles nullability.

04. Learn about Kotlin's null-safety operators and integration with Java types.

BY SIMON WIRTZ
SOFTWARE DEVELOPER

As Java developers, we're very accustomed to `NullPointerExceptions` (NPEs) that are thrown at the runtime of an application. This almost always happens unintentionally in consequence of a bug, which is based on unrecognized references to `null`. The `null` reference is often used to indicate absent values, which aren't obvious to the programmer in many cases. Although Java relies on strong static typing, it doesn't let you distinguish between reference types that can and cannot hold a `null` reference. Have a look at the following code example:

```java
java Device audio = new DeviceProvider().
getAudioDevice();
String audioName = audio.getName();
```

The method `getAudioDevice` returns an object of type `Device` but might return `null` in order to denote the absence of that device on particular systems. Well-documented methods will describe exactly that behavior, which still requires the developer to be very attentive. Not knowing about the possibility of a returned `null` reference is going to cause an awful `NullPointerException` in the subsequent call to `getName`. Wouldn't it actually be nice if we were able to identify the method's returned `Device` type as nullable (or non-nullable respectively) firsthand?

NULL-SAFETY IN JAVA

We have to find strategies that help us avoiding unwanted bugs due to NPEs. A common approach is to defensively check for `null` references and handle these cases in a way that makes more sense,

such as providing default values or throwing more meaningful exceptions. Applying this to the previous example brings us to the following solution:

```java
Device audio = new DeviceProvider().getAudioDevice();
String audioName;
if (audio != null) {
    audioName = audio.getName();
} else {
    throw new IllegalStateException("This system does
not provide an audio device.");
}
```

Wouldn't it actually be nice if we were able to identify the method's returned type as nullable (or non-nullable respectively) firsthand?

Constructs like these are part of any Java project and the approach works well unless you forget to add checks for certain variables.

It's a fairly error-prone approach and doesn't mitigate the fact that using null as a representative for absent things is risky.

Does Java offer any more sophisticated solutions to this problem? It does, at least in *some* situations. The Java SE 8 introduced the Optional type that acts as a "container object which may or may not contain a non-null value." This sounds very promising and needs to be considered next.

Instead of carelessly accessing that nullable type, the Optional type can be used for handling the possible nullability in various ways.

JAVA SE 8 OPTIONAL

The Optional type is a container that wraps another object, which can theoretically be null. Code that works on instances of Optional needs to handle the possible nullability in a rather explicit way:

```
Optional<Device> audio = new DeviceProvider().
getAudioDevice();
String audioName = audio
            .flatMap(Device::getName)
            .orElseThrow(() -> new
IllegalStateException("This system does not provide
an audio device."));
```

The getAudioDevice method was adjusted to return an Optional<Device>, which doubtlessly indicates to the client that the device can be absent. Instead of carelessly accessing that nullable type, the Optional type can be used for handling the possible nullability in various ways. These include providing default values and throwing exceptions with the help of simple methods like orElse and orElseThrow respectively. Furthermore, it literally forces the client to think about the potential null case.

Unfortunately, the whole Optional story already ends with this use case very suddenly. As stated by Java language architect Brian

Goetz in this StackOverflow post, the Optional type was not intended as a "general purpose Maybe [...] type" but a way to let libraries and APIs express the absence of a return type (as we saw in the previous example).

The Optional type is a great way to provide more explicit APIs that let the corresponding callers know exactly when null handling is required just by observing the method's signature. Nevertheless, it's not a holistic solution since it isn't meant to replace each and every null reference in the source code. Aside from that, can you safely rely on method return types, which are not marked as Optional?

NULL-SAFETY IN KOTLIN

After we have seen the rather unsafe null handling in the Java language, this section will introduce an alternative approach: The Kotlin programming language, as an example, provides very sophisticated means for avoiding NullPointerExceptions.

The language's type system differentiates between nullable and non-nullable types and every class can be used in both versions. By default, a reference of type String cannot hold null, whereas String? allows it. This distinction on its own doesn't make a very big difference, obviously. Therefore, whenever you choose to work with nullable types, the compiler forces you to handle possible issues, i.e. potential NPEs, appropriately.

```
//declare a variable with
//nullable String type, it's OK
//to assign `null` to it
var b: String? = "possiblyNull"
//1. does not compile, could throw NPE
val len = b.length
//2. Check nullability before access
if (b != null){
    b.length
}
//3. Use safe operator
val len = b?.length
```

This code example shows different ways of working with nullable types (String? in this case). As demonstrated first, it's not possible to access members of nullable types directly since this would lead to the same problems as in Java. Instead, traditionally checking whether that type is *not* null makes a difference. This action persuades the compiler to accept invocations on the variable (inside the if-block) as if it were not nullable. Note that this does

not work with mutable `var`s because they could possibly be set to `null` from another thread between the check and first action in the block.

As an alternative to explicit checks, the safe call operator `?.` can be used. The expression `b?.length` can be translated to "call `length` on `b` if `b` is not `null`, otherwise return `null`." The return type of this expression is of type `Int?` because it may result in `null`. Chaining such calls is possible and very useful because it lets you safely omit a great number of explicit checks and makes the code much more readable:

```
person?.address?.city ?: throw
IllegalStateException("No
city associated to person.")
```

Another very useful operator is the elvis operator `?:` that perfectly complements the safe call operator for handling `else` cases. If the left-hand expression is not `null`, the `elvis` operator returns it; otherwise, the right-hand expression will be called.

One of the key attributes of Kotlin is its fantastic interoperability with Java source code.

The last operator that you need to know is called not-null assertion operator `!!`. It converts any reference to a non-null type. This unchecked conversion may cause an NPE if that reference is `null` after all. The not-null assertion operator should only be used with care:

```
person!!.address!!.city //NPE will be thrown if
person or address is null
```

In addition to the shown operators, the Kotlin library provides plenty of helpful functions like `String?::IsNullOrEmpty()`, `String::toDoubleOrNull()`, and `List::filterNotNull()`, to name just a few. All of them support the developer in proper nullability handling and make NPEs almost impossible.

INTEROP BETWEEN BOTH LANGUAGES

One of the key attributes of Kotlin is its fantastic interoperability with Java source code. You can easily mix both languages in a project and call Kotlin from Java and vice versa. How does that work in the case of null safety, though?

As learned earlier in this article, every Java reference can be `null`, which makes it hard for Kotlin to apply its safety principles to them meaningfully. A type coming from Java is called platform type, denoted with an exclamation mark, e.g. `String!`. For these platform types, the compiler isn't able to determine whether it's nullable or not due to missing information. As a developer, when a platform type is, for example, assigned to a variable, the correct type can be set explicitly. If you assign a platform type `String!` to a variable of the non-nullable type `String`, the compiler allows it and as a consequence, safe access to that variable isn't being enforced. Nevertheless, if that decision turns out to be wrong, i.e. a `null` reference is returned, NPEs will be thrown at runtime. Fortunately, there's a solution that allows providing more information to the Kotlin compiler by applying certain annotations to the corresponding Java methods. This enables the compiler to determine actual nullability information and makes platform types unneeded.

BOTTOM LINE

It's important to understand that Kotlin does not try to avoid `null` references as such but raises the attention for `null`-related issues and especially NPEs enormously. If you take care of platform types and defensively decide to use them as nullable ones or apply the mentioned annotations to your Java libraries, you should be safe. `NullPointerExceptions` will be a thing of the past. As set out above, the Kotlin language provides many useful operators and other functions that simplify working with nullable types tremendously by making use of its clever type system. We can hope to see similar solutions in future Java versions soon.

SIMON WIRTZ is having fun as a Software Developer with a focus on Java (web-)applications for almost six years now. Currently, he works for a German software company in very different areas, mostly involved in Spring and OSGi development with Java. He started supporting Kotlin in January 2017 by blogging and speaking about it and also uses the language whenever possible.

DIVING DEEPER

INTO JAVA

#JAVA TWITTER ACCOUNTS

@s1m0nw1   @starbuxman   @trisha_gee   @mreinhold   @shelajev

@jboner   @myfear   @mariofusco   @javinpaul   @AdamBien

JAVA ZONES

Java

dzone.com/java

The largest, most active Java developer community on the web. With news and tutorials on Java tools, performance tricks, and new standards and strategies that keep your skills razor-sharp.

Microservices

dzone.com/microservices

The Microservices Zone will take you through breaking down the monolith step-by-step and designing microservices architecture from scratch. It covers everything from scalability to patterns and anti-patterns. It digs deeper than just containers to give you practical applications and business use cases.

Web Dev

dzone.com/webdev

The Web Dev Zone is devoted to all things web development—and that includes everything from front-end user experience to back-end optimization, JavaScript frameworks, and web design. Popular web technology news and releases will be covered alongside mainstay web languages.

JAVA REFCARDZ

Getting Started With Kotlin

Kotlin has become one of the most popular JVM languages in the past few months, partly because it experienced a lot of attention in the Android community after Google made Kotlin an official language for Android development. Download this Refcard to start working with this open-source language, which can be found on GitHub.

Core Java Concurrency

This Refcard will help Java developers working with multi-threaded programs to understand core concurrency concepts and how to apply them. Overview the key aspects of the Java language and get references on the core library.

Microservices in Java

This Refcard turns concepts into code and lets you jump on the design and runtime scalability train right away – complete with working Java snippets that run the twelve-factor gamut from config to service registration and discovery to load balancing, gateways, circuit breakers, cluster coordination, security, and more.

JAVA PODCASTS

Java Pub House

Learn about real issues in the Java world like O/R steups, threading, getting certain components on the screen, and troubleshooting.

Enterprise Java Newscast

Get the latest headlines, trends, and technologies in the world of enterprise software development.

The Changelog

Learn about topics like Ruby, Node.js, JavaScript, CSS, Git, and npm.

JAVA BOOKS

Clean Architecture

Learn how you can apply the universal rules of software architecture to dramatically improve developer productivity throughout the life of any software systems.

Making Java Groovy

Learn how to blend Groovy into your day-to-day work with Java.

Hibernate Tips

Learn how to efficiently implement your persistence layer with Hibernate's basic and advanced features.

ITEXT

Blockchain & PDF
the future of document security

PDF documents

Register

Verify

Properties

File storage

Database

ECM

ERP

iText pdfChain
PDF-Blockchain application layer

↔

Blockchain technology
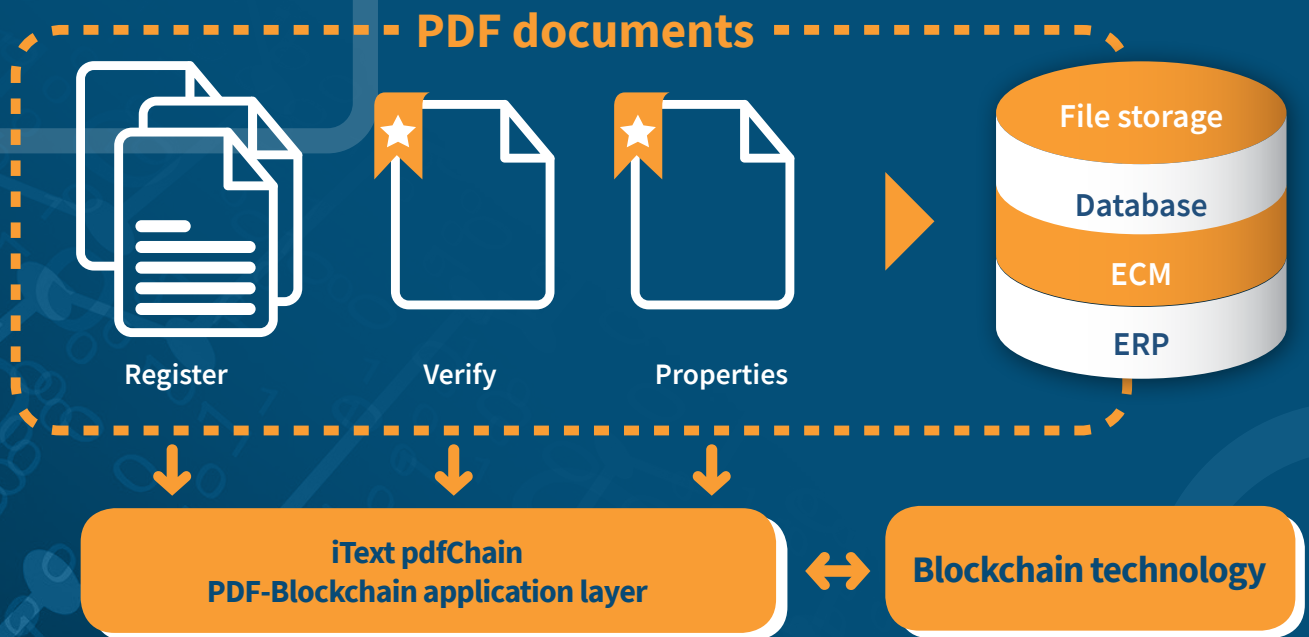
✔ **Non-sequential signing**

✔ **Decentralized database**

✔ **Secure certification**

✔ **Software agnostic**

Blockchain offers the opportunity to improve document security.
This can benefit a wide variety of markets including business, shipping, manufacturing
and more by decentralizing the authority authenticating documents. iText is working
on an open source version for PDF documents.

Learn more at :
→ **itextpdf.com/blockchain**

Is blockchain the future of document security? We think so.

Blockchain is the latest disruptive technology on the market. You may think of it as a synonym for "bitcoin," but the technology is useful in many more ways. Blockchain is a decentralized and distributed database that can solve many security issues with PDF documents. You can use this for recording events, managing records, provenance tracking, and document lifecycle management. We see opportunities to improve document security and use such as non-sequential signing, easily check the latest document version, and secure certification, all while being software-agnostic.

How can Java developers benefit from blockchain?

You can use our new FREE and open-source blockchain for documents. Easily integrate blockchain into your document security with our simple API. We provide an abstraction layer to simplify the link between PDF and blockchain technology, defining an API for registering, searching, and retrieving information from the blockchain. See our open-source blockchain project and consider how blockchain could help you track documents!

Aren't you a PDF company?

Yes, iText delivers well-known and proven technology for PDF creation, processing, and manipulation. Our technology is by developers for developers and offers a set of building blocks to easily integrate PDF functionality into your applications and workflows. Although we focus on PDF technology, our expertise and interest extend beyond it to help support digital workflows including digital signatures, long-term validation, document security, and document workflows.

WRITTEN BY RAF HENS
DIRECTOR OF PRODUCT MANAGEMENT, ITEXT SOFTWARE

PARTNER SPOTLIGHT

Blockchain for documents, the future of document security

"You may think of Blockchain as a synonym for "bitcoin," but the technology is useful for document workflows, too."

ITEXT

CATEGORY	OPEN SOURCE?	NEW RELEASE
New technology	Yes	Quarterly releases for commercial products and continuous updates for open-source products

CASE STUDY

Blockchain for documents, a use case:

Typical solutions for document security are vendor locked, but with blockchain you can use any software and leverage the core attributes of blockchain technology. A use case would include:

- Identifying the document on the blockchain with a hash or document ID
  - The hash value ensures the document cannot be modified
- Move signatures outside of the document onto the blockchain
  - Now you can use non-sequential signing
- Automate your transactions with smart contracts
  - Set up workflows in advance based on your needs, for example, if anyone pays X amount to Y account, with this text, automatically start Z transaction
- Set up your own cryptographic envelope (or you can still use a default)
- Create a blockchain-driven web of trust

STRENGTHS

- Track your documents
- Know if your document was forged or altered
- Validate other parties in your workflow

NOTABLE CUSTOMERS

- iText has serves more than half of the Fortune 50, SMEs, and pet projects around the world.

WEBSITE itextpdf.com	**TWITTER** @iText	**BLOG** itextpdf.com/posts/all

Java Collections Are Evolving

BY TRISHA GEE
DEVELOPER ADVOCATE, **JETBRAINS**

QUICK VIEW

01. Java 9 introduced a succinct new way to create immutable Lists, Sets, and Maps.

02. Java 10 added the ability to easily create immutable copies of Lists, Sets, and Maps.

03. In Java 10, it's now possible to collect the output of a stream operation into a collection that cannot be modified.

04. The Java REPL (JShell), added in Java 9, is a quick and easy way to experiment with code and see the output of various operations.

It's very easy as a developer to get into the routine of using the same tools the same way as always, particularly if the language we use has seen lengthy periods of extreme stability. Java is no longer in that mode: with releases every six months, we can expect to see updates to it very frequently. These updates may be large structural changes (like the Java Module System or API updates (for example, the Process API). Each release is likely to add something that's useful to us, so staying up-to-date is going to help us to be more productive developers.

In this article, we're going to specifically look at the updates to Collections in Java 9 (released September 2017) and Java 10 (released March 2018). Java Collections are a core part of the language that we probably touch in one way or another every day, and anything that makes it easier to work with Collections makes our job easier.

JAVA 9: CONVENIENCE FACTORY METHODS FOR COLLECTIONS

Java 9 introduced new ways to create immutable collections. At some point, we've all written code that looks something like this:

```
List<String> moods = Arrays.asList("HAPPY",
"SAD");
```

As of Java 9, you can now write the following instead:

```
List<String> moods = List.of("HAPPY", "SAD");
```

While saving six characters may be exciting to those who prefer very terse code, this might not seem like a huge improvement.

What's important to realize, though, is that this second case creates an immutable list.

We may have fallen into the trap of thinking `Arrays.asList` returns an immutable list because it's not possible to append to it:

```
jshell> List<String> moods = Arrays.
asList("HAPPY", "SAD");
moods ==> [HAPPY, SAD]

jshell> moods.add("ANGRY")
|  java.lang.UnsupportedOperationException thrown
|       at AbstractList.add (AbstractList.java:153)
|       at AbstractList.add (AbstractList.java:111)
|       at (#2:1)
```

However, changes are allowed within the limits of the list:

```
jshell> moods.set(0, "ANGRY")
$3 ==> "HAPPY"

jshell> System.out.println(moods)
[ANGRY, SAD]
```

It's likely this was not intended when we wrote this code. Instead, what we really wanted was:

```
List<String> moods =
Collections.unmodifiableList(Arrays
asList("HAPPY", "SAD"));
```

Now you can see that:

```
List<String> moods = List.of("HAPPY", "SAD");
```

...which does return an immutable list, is not only much shorter than the pre-Java-9 equivalent but also more correct than simply using `Arrays.asList`.

This is useful for Lists, but what's even more useful is that similar methods have been added for Sets and Maps. To create an immutable Set before Java 9, you could use something like:

```
Set<String> moods = Collections.
unmodifiableSet(new
HashSet<>(Arrays.asList("HAPPY", "SAD")));
```

This really was unpleasant, even when using static imports to reduce some of the noise. In Java 9, it's simply:

```
Set<String> moods = Set.of("HAPPY", "SAD");
```

There are also factory methods for creating Maps. Before Java 9, if we wanted to create a Map with a fixed set of values, we'd have to do something a little long-winded:

```
Map<String, Mood> wordToMood = new HashMap<>();
wordToMood.put("happy", HAPPY);
wordToMood.put("good", HAPPY);
wordToMood.put("great", HAPPY);
//… more values
wordToMood.put("horrible", SAD);
wordToMood.put("bad", SAD);
wordToMood.put("awful", SAD);
```

This was even worse if we wanted to initialize this for a constant (e.g. a static field), as it would have to be put somewhere in a static block, and wrapping this in an `unmodifiableMap` simply adds to the noise. In Java 9, this can be:

```
Map<String, Mood> wordToMood
  = Map.ofEntries(Map.entry("happy", HAPPY),
                  Map.entry("good", HAPPY),
                  Map.entry("great", HAPPY)
                  //...more values
                  Map.entry("horrible", SAD),
                  Map.entry("bad", SAD),
                  Map.entry("awful", SAD));
```

Static imports can make this more succinct, too. This `Map.ofEntries` method works for any arbitrary number of key/value pairs, as each pair is wrapped in a `Map.entry` and the `ofEntries` method takes a vararg. If the Map has fewer than ten values, we might want to use the convenience `Map.of` method, which takes up to ten key/value parameters:

```
Map<String, Mood> wordToMood = Map.of("happy", HAPPY,
                                      "good", HAPPY,
                                      "great", HAPPY,
                                      "horrible", SAD,
                                      "bad", SAD,
                                      "awful", SAD);
```

JAVA 10: CREATING IMMUTABLE COPIES OF COLLECTIONS

Java 9 introduced these factory methods to make it easier to create new immutable collections from known values. Java 10 recognizes that this is not the only way we create collections, and introduces more ways of creating immutable collections from existing collections or operations.

There's now an easy way to create an immutable Collection that's a copy of an existing Collection. Prior to Java 10, you could create a new list that was a copy of an existing collection using a copy constructor:

```
List<String> newCopyOfCollection = new
ArrayList<>(moods);
```

In this case, even if the original collection moods is immutable/unmodifiable, the new collection is not:

```
jshell> List<String> newCopyOfCollection = new
ArrayList<>(moods);
newCopyOfCollection ==> [HAPPY, SAD]

jshell> newCopyOfCollection.add("ANGRY")
$5 ==> true

jshell> System.out.println(newCopyOfCollection)
[HAPPY, SAD, ANGRY]
```

To create a list that cannot be changed, it would have to be wrapped in an unmodifiable list:

```
jshell> List<String> newCopyOfCollection =
Collections.unmodifiableList(new
ArrayList<>(moods));
newCopyOfCollection ==> [HAPPY, SAD]

jshell> newCopyOfCollection.add("ANGRY")
|  java.lang.UnsupportedOperationException thrown
|      at Collections$UnmodifiableCollection.add
(Collections.java:1056)
|      at (#8:1)
```

For Lists specifically, you can also use `Collections.copy`, but the syntax is a bit clunky and it's easy to get runtime errors if your destination list isn't set up correctly.

In Java 10, it's much easier to create a new immutable list from an existing Collection:

```
List<String> newCopyOfCollection = List.
copyOf(moods);
```

As you'd expect, you can't add or remove elements or alter the items in the list:

```
jshell> List<String> newCopyOfCollection = List.
copyOf(moods);
newCopyOfCollection ==> [HAPPY, SAD]

jshell> newCopyOfCollection.add("ANGRY")
|   java.lang.UnsupportedOperationException thrown
|       at ImmutableCollections.uoe
(ImmutableCollections.java:71)
|       at
ImmutableCollections$AbstractImmutableList.add
(ImmutableCollections.java:77)
|       at (#21:1)
```

Similar methods exist for Set:

```
Set<String> setCopyOfCollection = Set.
copyOf(moods);
```

...and Map. The Map version takes another Map to copy, not a Collection:

```
Map<String, Mood> copyOfMoodMap = Map.
copyOf(wordToMood);
```

From Java 10, you can create immutable Collections not only from some known values or by copying an existing Collection or Map, but also from Stream operations.

JAVA 10: CREATING IMMUTABLE COLLECTIONS FROM STREAMS

Java 10 makes it easy to create immutable collections from Stream operations, with the addition of the `toUnmodifiableList`, `toUnmodifiableSet`, and `toUnmodifiableMap` methods on Collectors. This means that from Java 10, you can create immutable Collections not only from

some known values or by copying an existing Collection or Map, but also from Stream operations.

Java is evolving to make it easier for us developers to write convenient and correct code.

For example, let's assume we have a Stream operation that takes a sentence and turns it into a list of unique words:

```
    List<String> uniqueWords = Pattern.compile("\\
s*[^\\p{IsAlphabetic}]+\\s*").splitAsStream(message)

.map(String::toLowerCase)

.distinct()

.collect(Collectors.toList());
```

This list is a simple `ArrayList` containing the results, and can be changed:

```
jshell> String message = "I am so so happy today,
and I am not happy every day";
message ==> "I am so so happy today, and I am not
happy every day"

jshell> List<String> uniqueWords = Pattern.
compile("\\s*[^\\p{IsAlphabetic}]+\\s*").
splitAsStream(message).
   ...>
map(String::toLowerCase).
   ...>                                   distinct().
   ...>
collect(Collectors.toList());
uniqueWords ==> [i, am, so, happy, today, and, not,
every, day]

jshell> uniqueWords.getClass()
$35 ==> class java.util.ArrayList

jshell> uniqueWords.add("SAD")
$36 ==> true

jshell> System.out.println(uniqueWords)
[i, am, so, happy, today, and, not, every, day, SAD]
```

If the aim of this stream operation was to return some fixed set of results, we probably didn't want to return an `ArrayList` but some sort of immutable list. In Java 10, we can do this:

```
List<String> uniqueWords
  = Patternc ompile("\\s*[^\\p{IsAlphabetic}]+\\s*"
      .splitAsStream(message)
      .map(String::toLowerCase)
      .distinct()
      .collect(Collectors.toUnmodifiableList());
```

This returns a List that cannot be changed:

```
jshell> List<String> uniqueWords =
Pattern.compile("\\s*[^\\p{IsAlphabetic}]+\\s*").
splitAsStream(message).
   ...>
map(String::toLowerCase).
   ...>
distinct().
   ...>
collect(Collectors.toUnmodifiableList());
uniqueWords ==> [i, am, so, happy, today, and, not,
every, day]

jshell> uniqueWords.getClass()
$39 ==> class java.util.ImmutableCollections$ListN

jshell> uniqueWords.add("SAD")
|   java.lang.UnsupportedOperationException thrown
|       at ImmutableCollections.uoe
(ImmutableCollections.java:71)
|       at
ImmutableCollections$AbstractImmutableList.add
(ImmutableCollections.java:77)
|       at (#40:1)
```

There's also `Collectors.toUnmodifiableSet`, which may be more appropriate in this scenario since the Collection contains only unique values.

`Collectors.toUnmodifiableMap` is for creating immutable Maps, and, like `toMap`, is a little trickier, as it means we need to give functions to define what the keys and the values are. If we change our example to use a Map to calculate the number of times each word is in the sentence, we can demonstrate how to collect into an immutable Map:

```
    Map<String, Long> wordCount = Pattern.
compile("\\s*[^\\p{IsAlphabetic}]+\\s*").
splitAsStream(message).

map(String::toLowerCase).
```

CODE CONTINUED ON NEXT COLUMN

```
collect(Collectors.toUnmodifiableMap(Function.
identity(),

word -> 1L,

(oldCount, newVal) -> oldCount + newVal));
```

As before, we can see that values can't be added to or removed from this Map:

```
jshell> Map<String, Long> wordCount =
Pattern.compile("\\s*[^\\p{IsAlphabetic}]+\\s*").
splitAsStream(message).
   ...>
map(String::toLowerCase).
   ...>
collect(Collectors.toUnmodifiableMap(Function.
identity(),
   ...>
word -> 1L,
   ...>
(oldCount, newVal) -> oldCount + newVal));
wordCount ==> {and=1, i=2, am=2, day=1, so=2,
every=1, today=1, not=1, happy=2}

jshell> wordCount.getClass()
$49 ==> class java.util.ImmutableCollections$MapN

jshell> wordCount.put("WORD", 1000L)
|   java.lang.UnsupportedOperationException thrown
|       at ImmutableCollections.uoe
(ImmutableCollections.java:71)
|       at
ImmutableCollections$AbstractImmutableMap.put
(ImmutableCollections.java:558)
|       at (#50:1)
```

In conclusion, we can see that Java is evolving to make it easier for us developers to write convenient and correct code. Some of these changes are just small additions to existing APIs, so it's easy to miss them in the fuss of bigger language changes. Collections have evolved in the most recent versions of Java, and if we stay up-to-date and use these changes, we'll find our lives a little bit easier.

TRISHA GEE has developed Java applications for a range of industries, including finance, manufacturing, software and non-profit, for companies of all sizes. She has expertise in Java high performance systems, is passionate about enabling developer productivity, and dabbles with Open Source development. Trisha is a leader of the Sevilla Java User Group and a Java Champion.

Local Variable Type Inference in Java

QUICK VIEW

01. JEP 286 has been introduced in JDK 10 and finally ushers in type inference for initialized local variables.

02. Although the use of var can result in diminished readability, a large portion of the readability of var hinges on the effective naming of variables.

03. Explicit typing can still be used, but var provides more options to developers and steers Java towards the conciseness of many modern programming languages

BY JUSTIN ALBANO
SOFTWARE ENGINEER, **CATALOGIC SOFTWARE**

Variable type inference — the ability of the compiler to automatically deduce the type of a variable — has become an essential part of most strongly typed programming languages. While many languages such as C++, Scala, Kotlin, and C# have introduced rich type inference mechanics over the years, Java has been noticeably absent in this department. Prior to JDK 10, a vast majority of the type inference capability of Java was focused on the right-hand side (RHS) of expressions and was largely seen in lambda argument type inference, the diamond operator, and generic method argument type inference:

```
    listOfStudents.removeIf(s -> s.getId() ==
desiredId);
    List<Integer> ints = new ArrayList<>();

public <T> Bar foo(List<T> elements) { /* … */ }
Bar bar = foo(new ArrayList<Integer>());
```

Although these features eliminated a great deal of development tedium, Java has been severely lacking left-hand side (LHS) type inference support — namely, variable type inference. With the adoption of JDK Enhancement Proposal (JEP) 286 in JDK 10, Java now includes the var reserved type name which allows for the LHS type (known as the manifest type) of initialized local variable to be inferred by the Java compiler:

```
    var i = 0;
    var students = new ArrayList<Student>();
    var iter = list.listIterator();
```

This may seem to be a trivial improvement in the language, but its true usefulness becomes evident when comparing the following declarations:

```
    Iterator<List<SomeLongBeanName>> iterator =
someList.iterator();
    var iterator = someList.iterator();
```

The ability of the compiler to deduce the type of a variable removes much of the clutter in code that is otherwise overly verbose. It is important to note that var is reserved type name and *not* a keyword. This means that var can still be used as a variable name, method name, or package name, but cannot be used as a class or interface name. Thus, the following are all *valid* uses of var:

```
    var var = 0;
    public void var(int var) { /* … */ }
    package var;
```

Since var is treated as a reserved type name, classes and interfaces can no longer be named var since the compiler would be unable to distinguish between a variable of type var or the use of type inference. This creates a backwards incompatibility with existing JDK releases, but the use of

var as a class or interface name is counter to long-held Java conventions and therefore, it is unlikely that such a class or interface is currently in use.

WHERE CAN VAR BE USED?

Although some languages allow for very powerful type inference schemes, the Java team decided to reduce the scope and applicability of the var reserved type to a few specific contexts:

- Local variables with initializers

- Indices in the enhanced for-loop

- Locals declared in traditional for-loops

Therefore, all of the following are *valid* uses of var as a reserved type name:

```
var foo = 1;
for (var student: students) { /* … */ }
for (var i = 0; i < 10; i++) { /* … */ }
```

This means that var cannot be used as the type of method parameters, constructor parameters, method return types, catch parameters in try-catch statements, fields, or any other type of variable declaration. Although this may appear restrictive, the three valid contexts enumerated above cover a large number of practical cases without requiring complex type inference mechanics.

Although some languages allow for very powerful type inference schemes, the Java team decided to reduce the scope and applicability of the var reserved type to a few specific contexts.

In many cases, a developer expects the compiler to read his or her mind in order to infer the correct type of a variable or parameter. For example, what is the correct type of a variable initialized to null? Or a lambda expression that accepts a single argument and returns a value of the same type? What's more, how should type inference be handled

if a variable is not initialized at declaration and assigned a value at a later time, possibly in a distant portion of code?

Being that developers must have a solid understanding of how types are inferred (in order to be comfortable allowing the compiler to infer types) and type inference is no trivial matter for compilers, the Java team settled on a very simple implementation of type inference: A single reserved type name var that handles only local variables and for-loop locals.

The Java team settled on a very simple implementation of type inference: A single reserved type name var that handles only local variables and for-loop locals.

WHY VAR?

It is important to note that a great deal of research was performed in selecting var as the only type inference mechanism thus far. Many languages make a distinction between immutable variable initialization (i.e. `final int x = 0`) and mutable variable initialization (i.e. `int x = 0`) and use different type inference qualifiers for each. For example, Scala and Kotlin both use var and val to denote mutable and immutable variable initialization, respectively. While differing type names do provide an important visual distinction between these initializations, it also disrupts the uniformity of variable declarations:

```
var mutableX = …
var mutableY = …
val immutableZ = …
var mutableA = …
```

In order to resolve this fiercely subjective issue, a survey was released by the JEP 286 team in 2016 that inquired of developers which scheme was most desirable. It found that 24% of respondents preferred var only, 48% preferred the combination of var and val, and 14% preferred the combination of var and let (like in Swift). Although the var-val pairing appeared to win the contest, the survey also found that 80% of respondents were accepting of var

only and 19% hated var only, 74% accepted and 26% hated the var-val pairing, and 59% accepted and 41% hated the var-let pairing. Thus, the Java team decided on the most acceptable and least hated solution: var only.

Reserved Type Name Preference
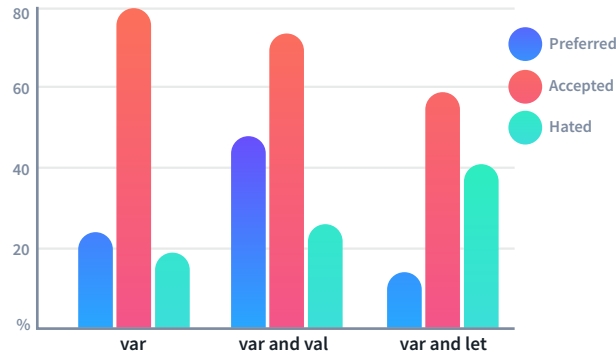
While 74% of survey respondents favored the inclusion of local variable type inference (12% more were accepting and only 10% thought it was a bad idea), one of the most ardent, albeit justifiable, complaints of variable type inference in Java is reduced readability.

The release of JDK 10 has finally ushered in local variable type inference in the form of var.

WHAT ABOUT READABILITY?

One of the most repeated complaints local variable type inference in Java is its uncanny ability to make code less readable. One of the most used examples of this drawback is the following:

```
var x = y.getFoo();
```

While there is some merit to this complaint, it is not completely justified. As pointed out by the Brian Goetz, this lack of readability is derived primarily from the lackluster naming of the variable x, not from its hidden type. This example demonstrates that readability has much more to do with variable naming than the explicit typing of variables. Therefore, with proper variable naming, var can actually improve readability in some very important ways, as seen in the following snippet:

```
Map<Integer, Student> idToStudentMap =
epository.getStudentIdMap();
List<Student> enrolledStudents = repository.
netEnrolledStudents();
Address addressOfTopStudent = topStudent.
getAddress();

var idToStudentMap = repository.
getStudentIdMap();
var enrolledStudents = repository.
getEnrolledStudents();
var addressOfTopStudent = topStudent.
getAddress();
```

Apart from removing visual clutter, the use of var also improves the focus on the variable names in variable declarations. As with all features of a language, var should be used with care. Using var indiscriminately will likely reduce the readability of code, but there are many cases in which the opposite is true: Using var diligently will likely improve the readability of code. For example:

```
for (Iterator<String> it = list.iterator();
it.hasNext(); ) {
System.out.println(it.next());
}

for (var it = list.iterator(); it.hasNext(); ) {
System.out.println(it.next());
}
```

CONCLUSION

Java has been notoriously lagging in its type inference support, but the release of JDK 10 has finally ushered in local variable type inference in the form of var. While var is noticeably simpler than its counterpart in other languages, with proper diligence, it can improve the conciseness of code while reducing the tedium of explicit typing. Paired with future improvements to lambda type inference (JEP 323) and enum types (JEP 301), the addition of var is a key part in moving Java from a infamously verbose language to a reputably terse language.

JUSTIN ALBANO is a Software Engineer at Catalogic Software, Inc. responsible for building distributed catalog, backup, and recovery solutions for Fortune 50 clients, focusing on Spring-based REST API and MongoDB development. When not working or writing, he can be found at the ice rink, watching hockey, drawing, or reading.

LOOKING AT JDK 10 AND JDK 11

WRITTEN BY DUSTIN MARX

JDK 10

SUMMARY JAVADOC TAG

JDK 10 introduces a new Javadoc tag `{@summary}` via issue JDK-8173425 ("Javadoc needs a new tac to specify the summary."). This new tag allows developers to explicitly specify what portion of the Javadoc comment appears in the "Summary" rather than relying on Javadoc's default treatment looking for a period and a space to demarcate the end of the summary portion of the comment. `{@summary}` allows for explicit control of what appears in the method's summaries.

The following code demonstrates `{@summary}` in Javadoc method comments.

```
package dustin.examples.javadoc;
/**
 * Demonstrate JDK 10 added summary support.
 * Demonstrates this by comparing similar methods'
 * Javadoc comments with and without use of new
 * "@summary" tag.
 */
public class Summary
{
   /**
    * This method's first sentence is normally
    in the summary.
    * Here are some of its characteristics:
    * <ul>
    * <li>This method does great things.</li>
    * <li>This method does not really do
      anything.</li>
    * </ul>
    */
```

CODE CONTINUED ON NEXT COLUMN

```
public void implicitSummary1()
{
}
/**
 * This method's first sentence is normally in
   the summary.Here are some of its
   characteristics:
 * <ul>
 * <li>This method does great things.</li>
 * <li>This method does not really do
   anything.</li>
 * </ul>
 */
public void implicitSummary2()
{
}
/**
 * {@summary This method's first sentence is
   normally in the summary. Here are some of
   its characteristics:
 * <ul>
 * <li>This method does great things.</li>
 * <li>This method does not really do
   anything.</li>
 * </ul>}
 */
public void explicitSummary1()
{
}
/**
 * {@summary This method's first sentence is
   normally in the summary.Here are some of its
   characteristics:
 * <ul>
 * <li>This method does great things.</li>
 * <li>This method does not really do
   anything.</li>
 * </ul>}
 */
public void explicitSummary2()
{
}
}
```

ACCESSING A JAVA APP'S PROCESS ID FROM JAVA

JDK 10 introduces an easy approach to obtaining a JVM process's PID via a new method on the `RuntimeMXBean`. JDK-8189091 ("MBean access to the PID") introduces the `RuntimeMXBean` method `getPid()` as a default interface method with JDK 10. The following code demonstrates the use of the new `getPid()` method on `RuntimeMXBean`.

```
final RuntimeMXBean runtime =
ManagementFactory.getRuntimeMXBean();
final long pid = runtime.getPid();
final Console console = System.console();
out.println("Process ID is '" + pid +
"'Press <ENTER> to continue.");
console.readLine();
```

FUTURETASK GETS A TOSTRING()

The JDK class FutureTask, introduced with J2SE 5, finally gets its own `toString()` implementation in JDK 10. The addition of a specific implementation of `toString()` to the `FutureTask` class in JDK 10 is a small one. However, to developers "staring at output of `toString` for 'task' objects (Runnables, Callables, Futures) when diagnosing app failures," as described in JDK-8186326's "Problem" statement, this "small" addition is likely to be very welcome.

JDK 11

FIVE NEW JEPS TARGETED FOR JDK 11

Five new JEPs will bring the number of JEPs currently associated with JDK 11 to a total of eight. They are JEP 324: Key Agreement With Curve 25519 and Curve448, JEP 327: Unicode 10, JEP 328: Flight Recorder, JEP 329: ChaCha20 and Poly1305 Cryptographic Algorithms, and JEP 330: Launch Single-File Source-Code Programs.

1. JEP 324

The primary goal of JEP 324 is to provide an API and an implementation for the RFC 7748 standard. This particular elliptic curve is well-suited as an addition to the JDK that offers 128 bits of security and is one of the fastest ECC curves.

2. JEP 327

JEP 327 aims to "upgrade existing platform APIs to support version 10.0 of the Unicode Standard." However, it will not include four related Unicode specifications: UTS #10 ("Unicode Collation Algorithm"), UTS 39 ("Unicode Security Mechanisms"), UTS #46 ("Unicode IDNA Compatibility Processing"), and UTS 51 ("Unicode Emoji").

3. JEP 328

JEP 328 provides "a low-overhead data collection framework for troubleshooting Java applications and the Hotspot JVM." It allows for analyzing issues in the period leading up to a problem by recording events and storing them "in a single file that can be attached to bug reports and examined by support engineers."

4. JEP 329

JEP 329 aims to replace "the older, insecure RC4 stream cipher" with the "relatively new stream cipher" known as ChaCha20 that is currently considered secure.

5. JEP 330

JEP 330 will allow "the Java launcher to run a program supplied as a single file of Java source code." This makes it easier to write Java-based scripts and is intended to help developers new to Java start applying the language syntax more quickly.

NEW METHODS ON JAVA STRINGS
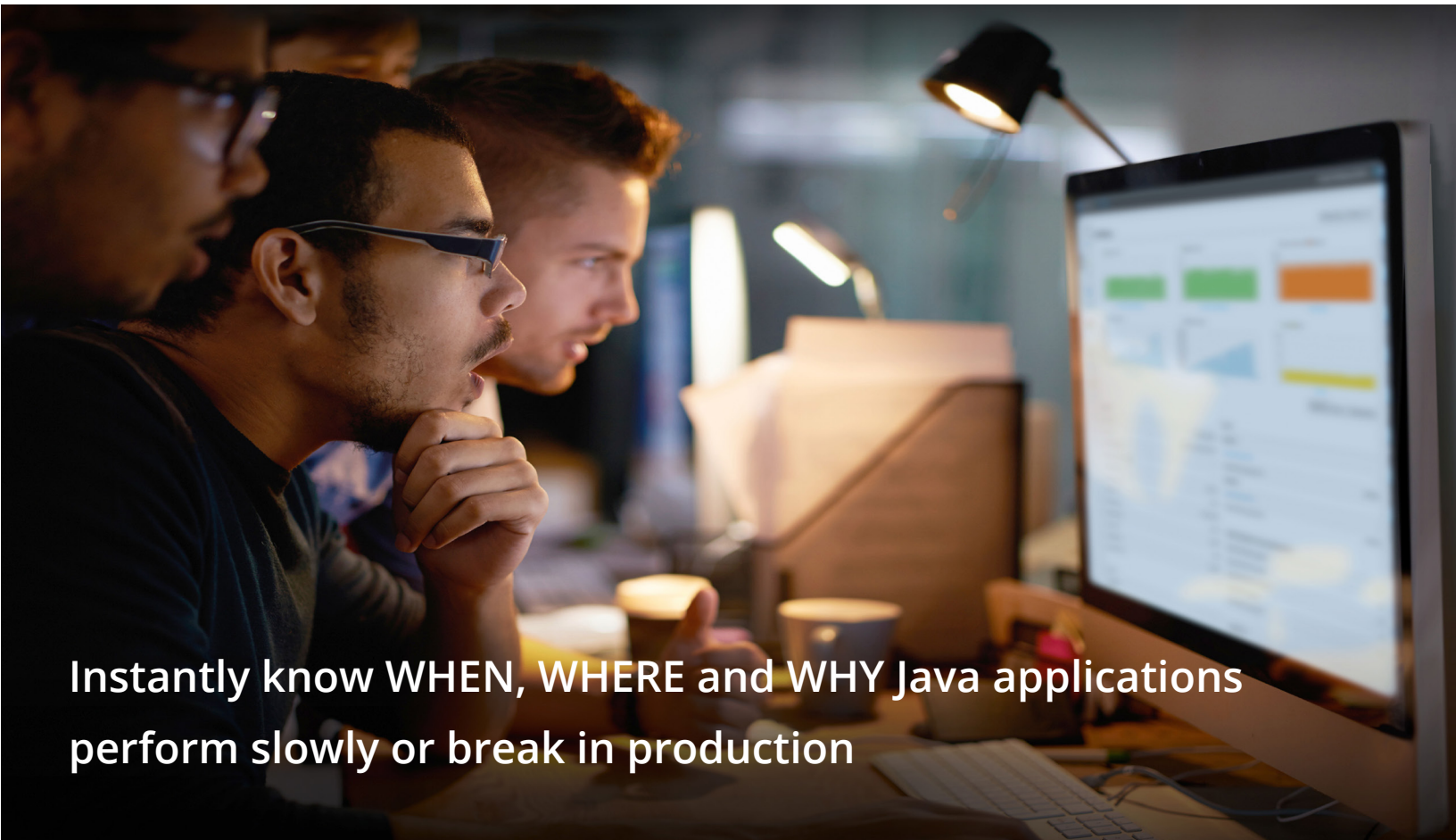
The new methods on String currently proposed for JDK 11 provide a more consistent approach to handling white space in strings that can better handle internationalization, provide methods for trimming white space only at the beginning of the string or at the end of it, and provide a method especially intended for coming raw string literals. Evidence of the progress that has been made related to these methods can be found in messages requesting "compatibility and specification reviews" (CSR) on the core-libs-dev mailing list.

NEW METHODS ON JAVA FILES

New methods are currently planned for the `java.nio.file.Files` class that will allow for reading a file's content into a String, writing a String into a file, and determining whether two files have the same content.

NEW PREDICATE NEGATION

A static function `Predicate.not()` is planned for JDK 11 that will make it easier to negate predicate lambda expressions.

fusion reactor ™

Instantly know WHEN, WHERE and WHY Java applications perform slowly or break in production

FusionReactor goes beyond traditional APM tools to give you unrivaled insight into how your Java code performs and executes in production

FusionReactor is the only solution that combines traditional application performance monitoring (APM) capability with developer focused profiling tools and a production safe, low overhead Java debugger.   Use FusionReactor across your whole software development life cycle (SDLC) and reduce the average time to identify and fix critical production application errors from days into minutes.

FusionReactor - Find it. Fix it. Prevent it.

Measure performance across the application

Get instant code level / variable state visibility

< 1 % overhead - swift, safe and secure to use

No code changes req'd deploy in minutes

Start Free Trial   www.fusion-reactor.com

How to cut issue isolation time down from days to minutes

This year's DZone Performance Guide reported that 36% of the developers interviewed, said that they had to solve performance problems "every single week." The survey also reported that 32 % of respondents said it takes between 11 and 30 hours to actually identify and solve the problem.

Why are performance problems increasing? One reason is that the focus of building functionality before performance is still paramount. Quite understandable, as businesses push for shorter release cycles and added functionality.

WHAT NEEDS TO CHANGE?

The process of reviewing performance needs to become a core part of the Software Development Life Cycle (SDLC) and when apps break or perform poorly in production, developers need real-time insight and transparency into what's happening at the point that things are actually breaking – in production.

FusionReactor offers a solution to both of these challenges. At its core, it's a monitoring tool and includes alerting plus all the instrumentation data you need to measure the performance of your application, DB queries and resources. However, unlike other monitors, it includes a fully integrated IDE style debugger and a suite of developer focused analysis tools, which enable you to perform real-time code profiling as well as heap and thread state analysis.

FusionReactor is designed to trade off production impact against production insight

FusionReactor does not provide all of the features you can find in mainstream developer tools such as YourKit, JProfiler or VisualVM – it was never meant to. Using as little as 1% of system resource, FusionReactor is designed to trade off production impact against production insight. The objective being to cut issue isolation time down from days to minutes.

Cut root cause analysis time down to size, join the revolution and 5,000+ other companies who are using and benefiting from what FusionReactor can offer.

WRITTEN BY DAVID TATTERSALL
CEO/CO-FOUNDER – INTERGRAL GMBH

FusionReactor

FusionReactor will instantly show you WHEN, WHERE, and WHY your Java applications perform slowly or are breaking in production

CATEGORY	OPEN SOURCE?	NEW RELEASE
APM for Developers and DevOps	No	Monthly

CASE STUDY

Bullhorn provides cloud-based CRM solutions for relationship-driven businesses. Its zero-click data capture technology and relationship intelligence gives companies what they need, from insight to action, to win new customers and keep them happy.

"FusionReactor allows our team at Bullhorn to respond to issues quickly before they become customer impacting. Its short polling interval gives us a needed edge when it comes to ensuring an excellent experience for our customers. FusionReactor outshines the competition with its rapid response time, small hardware footprint, and low total cost of ownership."

Brad Witherell - *Manager, Systems Engineering and Administration Bullhorn*

STRENGTHS

- Instantly isolate production performance and stability issues

- Deep dive, real-time monitoring and profiling for Java applications

- Integrated, low-overhead debugger, code, thread, & memory profiler

- Crash protection capability to increase application resilience

- Full featured monitoring capability - full alerting & metric analysis

- On-premise & SaaS version available plus 14-day free trial

NOTABLE CUSTOMERS

- Bullhorn
- Auto Europe
- Ford Motor Co.
- Foot Locker
- Hasbro

WEBSITE fusion-reactor.com   **TWITTER** @Fusion_Reactor   **BLOG** blog.fusion-reactor.com

Java EE 8 and the Open Future of Java EE

BY REZA RAHMAN
SENIOR VICE PRESIDENT, **AXIONIQ**

QUICK VIEW

01. A unique characteristic of Java EE 8 is that it has been one of the most community opinion-driven major technology release in the history of Java.

02. The goal of the new Java EE Security API is to make common, simple security needs portable by introducing embeddable authentication and authorization via annotations and CDI.

03. Oracle will be donating all Java EE code to the Eclipse Foundation.

Java EE 8 has been released and includes a significant amount of changes relevant to server-side Java developers. This article will overview these changes at a high level and look at some representative code examples.

A unique characteristic of Java EE 8 is that it has been one of the most community opinion-driven major technology releases in the history of Java. The scope of Java EE 8 was determined by two separate developer surveys: one conducted before Java EE 8 development started and one toward the end of when Java EE 8 was released. As a result, Java EE 8 is a very feature-complete release, particularly for applications that don't need fine-grained microservices features. Nonetheless, such features have already been brought to the Java EE ecosystem through the MicroProfile initiative and will likely be standardized as part of the next major release of Java EE as Jakarta EE 9.

THEMES

Let's briefly look at the distinguishable themes for Java EE 8 before diving into feature details.

- Web standards alignment
- HTTP/2, SSE, JSON
- CDI alignment
- CDI 2, JSF managed bean pruning, injecting JSF artifacts, CDI support in JPA
- Simplicity
- Security, EJB pruning
- Java SE 8 alignment
- JSF, JPA, JAX-RS, JMS, Bean Validation, JSON-P, CDI

SERVLET 4

The principal goal of Servlet 4 is to bring HTTP/2 support to server-side Java. HTTP/2 is a fundamental modernization of the protocol that keeps the internet together.

Because these are largely protocol layer changes, they can be transparently handled by the Servlet 4 runtime without any API changes. The Servlet 4 certification tests that a container properly implements HTTP/2. This is very important since no other certification process for HTTP/2 exists. Servlet 4 does introduce a simple API change to enable server-push. However, even this change is transparently absorbed at a lower level in the case of JSF 2.3 users.

JSON-B (JAVA API FOR JSON BINDING)

Making JSON a first-class citizen of the platform has been a goal since Java EE 7. Using JSON should not require installing or configuring yet another library. To that end, a low-level parsing API called JSON-P (Java API for JSON Processing) was introduced in Java EE 7. Java EE 8 introduces a higher-level JSON binding API called JSON-B that makes it feel like JSON is as native as Java serialization in Java EE. The idea is that converting POJOs to/from JSON should just work by default without needing to add any annotations. JSON-B does include a small number of annotations to override default mappings such as `@JsonbProperty` (to rename fields) and `@JsonbTransient` (for fields to be ignored by serialization). The example below shows these concepts in action.

```
@GET ...
@Produces("application/json")
public Person getPerson(...) {

    ...
    Person duke = new Person();
    duke.setName("Duke");
    duke.setGender("Male");
    phones = new HashMap<>();
    phones.put("home", "650-123-4567");
    phones.put("mobile", "650-234-5678");
    duke.setPhones(phones);

    return duke;
}
```

In the example above, the Person POJO does not have any JSON-B annotations. Because JAX-RS 2.1 integrates with JSON-B in Java EE 8, the Person POJO will be automatically converted to the JSON below in the HTTP response since the output is specified to be of type *application/json*.

```
{"name":"Duke",
 "gender":"Male",
 "phones":{
     "home":"650-123-4567",
     "mobile":"650-234-5678"}}
```

JSON-P 1.1

JSON-P was fairly feature-complete in Java EE 7. In Java EE 8 JSON-P incorporated updates in the JSON standards space such as JSON-Pointer, JSON-Patch, and JSON-Merge/Patch. JSON-Pointer allows for looking up values in a JSON structure using a URL-like path. JSON-Patch allows for issuing commands to modify parts of a JSON structure. JSON-Patch depends on JSON-Pointer to reference locations in a JSON structure. JSON-Merge/Patch is similar to JSON-Patch but offers capabilities to do sophisticated merges and diffs of JSON structures.

The best way to explore these features is through a simple example. Let's start with the following JSON structure:

```
[{"name":"Duke",
  "gender":"Male",
  "phones":{
      "home":"650-123-4567",
      "mobile":"650-234-5678"}},
 {"name":"Jane",
  "gender":"Female",
  "phones":{
      "mobile":"707-555-9999"}}]
```

The following shows how using two JSON-Patch commands to modify the above JSON structure looks. The first command updates the mobile phone number for "Duke". The second command removes "Jane" from the list of persons. /0/phones/mo-

bile and /1 are examples of JSON-Pointers. Aside from replace and remove, JSON-Patch supports operations like add, move, copy, and test.

```
[{"op": "replace", "path":"/0/phones/mobile",
 "value":"650-111-2222"},
  {"op": "remove", "path":"/1"}]
```

The following is how the JSON-P 1.1 code to apply these JSON-Patch operations looks. The target object holds the JSON-P structure for the persons array.

```
JsonPatchBuilder builder = new JsonPatchBuilder();
JsonArray result = builder
    .replace("/0/phones/mobile", "650-111-2222")
    .remove("/1")
    .apply(target);
```

SSE (SERVER-SENT EVENTS)

SSE is part of HTML5. SSE allows for server-to-client streaming of events over HTTP. Under the hood, SSE is a long-lived HTTP connection that uses a specialized content-type: text/event-stream. Events are typically distinct JSON objects sent from the server to the client over time. SSE is useful for "stock ticker"-type applications and monitoring consoles. SSE is supported both on the server and client side in JAX-RS 2.1. The following is a server-side example:

```
@Path("tickers")
public class StockTicker {
    @Resource ManagedExecutorService executor;

    @GET @Produces("text/event-stream")
    public void getQuotes(
            @Context SseEventSink sink,
            @Context Sse sse) {
        executor.execute(() -> {
            ...
            sink.send(sse.newEvent(stockqoute));
            ...
        });
    }
}
```

In the example, a browser connects to the server using the "tickers" endpoint. The endpoint produces a series of stock quote updates in a background thread and sends them to the client over an SSE event sink connection pipe using the Sse event builder utility.

JAVA EE SECURITY

The goal of the new Java EE Security API is to make common, simple security needs portable by introducing embeddable authentication and authorization via annotations and CDI. At a high level, three new features are introduced:

1. It is possible to specify through simple annotations whether the application uses basic, form-based, or custom authentication.

2. It is possible through simple annotations to specify that authentication and authorization data are stored in the database or LDAP directory. If the built-in identity stores are not enough, it is possible to have a simple CDI bean in the application act as an identity store.

3. A universal security context is made available through CDI injection. This context provides a handle to information about the currently logged-in user that can be used anywhere including in custom security interceptors. This is in addition to the existing `@RolesAllowed` annotation.

The following annotation example that specifies database security is illustrative of just how simple Java EE 8 security is.

```
@DataBaseIdentityStoreDefinition (
    dataSourceLookup="java:global/MyDB",
    callerQuery="SELECT password FROM principals
WHERE username=?",
    groupsQuery="SELECT role FROM roles where
username=?", ...)
```

CDI 2

One of the key changes in CDI 2 is the standardization of a bootstrap mechanism in plain Java SE environments. This has meant breaking CDI into three parts: core, Java SE, and Java EE. These changes enable CDI to be adopted by more technologies — inside and outside of Java EE. These changes have enabled CDI to be used as a core technology for the MicroProfile initiative. Another key change in CDI 2 is making events completely asynchronous. The following example shows the feature.

```
@Inject @CargoInspected Event<Cargo> cargoInspected;
...
public void inspectCargo(TrackingId trackingId) {
    ...
    cargoInspected.fireAsync(cargo);
}
public void onCargoInspected(
    @ObservesAsync @CargoInspected Cargo cargo) {
```

The `inspectCargo` method thread gets control back immediately after the `fireAsync` method is invoked. The `onCargoInspected` observer method is invoked on a completely separate thread. CDI 2 also made several simplifications to its extensibility APIs to further encourage the CDI plugin ecosystem. Lastly, CDI 2 adapts to Java SE 8 features such as lambdas, completable futures, streams, and repeatable annotations.

Beyond the changes in CDI 2 itself, a number of technologies improved their alignment with CDI in Java EE 8. For example, JSF 2.3 makes key artefacts like the `FacesContext` injectable and deprecates its own older managed bean model in favor of CDI. JPA 2.2 also improved its alignment with CDI by making artefacts like attribute converters injection-capable.

A key piece of CDI alignment had been slated for Java EE 8 through JMS 2.1. JMS 2.1 aimed at creating a CDI-based JMS listener to replace EJB message-driven beans. Similarly, EJB annotations like `@Asynchronous` and `@Schedule` could be made available to all CDI beans through the Java EE Concurrency Utilities. Unfortunately, Oracle decided to discontinue this work for Java EE 8. This is likely work that will be done in the next major Java EE revision. Fortunately, some work towards deprecating EJB was done in Java EE 8 such as pruning CORBA interoperability.

REPEATABLE ANNOTATIONS

Prior to Java SE 8, it was not possible to repeat annotations. As a result, where annotations needed to be repeated, Java EE used wrapper annotations, as shown below:

```
@NamedQueries({
    @NamedQuery(name=SELECT_ALL, query="..."),
    @NamedQuery(name=COUNT_ALL, query="...")})
public class Customer {
```

As of Java SE 8, such annotations have been adapted to be repeatable, as shown below.

```
@NamedQuery(name=SELECT_ALL, query="...")
@NamedQuery(name=COUNT_ALL, query="...")
public class Customer {
```

The technologies in Java EE 8 that have adapted to repeatable annotations include JPA, JMS, JavaMail, Bean Validation, EJB, and CDI.

DATE-TIME API

The new Java SE 8 date-time API is more feature-complete, easy-to-use, and internationalized compared to the older Java SE date handling functionality.

As the following example shows, both JPA 2.2 and Bean Validation 2 have been updated to natively support the date-time API.

```
@Entity
public class Accident {
    ...
    @Temporal(TemporalType.TIMESTAMP)
    @Past
    private Instant when;
    ...
}
```

JPA 2.2 knows how to properly read the when field of type `Instant` from the database and write it back. Similarly, all Bean Validation 2 annotations correctly validate date-time API types. JSF 2.3 can also correctly convert and validate all date-time types without any additional code.

A unique characteristic of Java EE 8 is that it has been one of the most community opinion-driven major technology releases in the history of Java.

COMPLETABLE FUTURE

Java SE 8 completable futures bring JavaScript promises to Java. Compared to the older Java SE `Future` interface, completable futures are non-blocking, lambda-friendly, and composable. These characteristics tend to be very important while asynchronously invoking interrelated RESTful endpoints that are part of a microservices-based system.

These are the reasons the JAX-RS 2.1 client API has been adapted to make use of completable futures. The following example shows the feature.

```
CompletionStage<Assets> getAssets = client
    .target("assets/{ssn}")
.resolveTemplate("ssn", person.getSsn())
.request("application/json")
    .rx()
    .get(Assets.class);
CompletionStage<Liabilities> getLiabilities = client
    .target("liabilities/{ssn}")

.resolveTemplate("ssn", person.getSsn())

.request("application/json")
    .rx()
    .get(Liabilities.class);
Coverage coverage = getAssets.
thenCombine(getLiabilities,
    (assets, liabilities) -> underwrite(assets,
liabilities))
        .toCompletableFuture().join();
```

STREAMS

Java SE 8 streams utilize lambdas to provide high-performance, concise aggregate operations on collections of objects. Operations include `filter`, `transform`, `sum`, `average`, `min`, `max`, and `sort`.

JPA 2.2 query results can now return streams. The following example shows the feature.

```
Stream<Book> books = entityManager.createQuery(
    "SELECT b FROM Book b", Book.class).
getResultStream();
books.map(b -> b.getTitle() + " was published on " +
b.getPublishingDate())
    .forEach(s -> log.info(s));
```

JSON-P 1.1 was also adapted to easily convert between JSON arrays and streams.

Aside from these headline features, Java EE 8 contains a lot more that you should look into yourself. For example, CDI 2 can now order events, JSF 2.3 provides an easy way to use WebSocket, JAX-RS 2.1 allows for broadcasting SSE events to multiple clients, Bean Validation 2 adds new constraints like `@Email`, `@NotEmpty`, `@NotBlank`, `@Positive`, `@Negative`, and much more.

MICROPROFILE, JAKARTA EE, AND THE FUTURE OF JAVA EE

The MicroProfile initiative was started at the Eclipse Foundation parallel to Java EE 8 to bring features required for fine-grained microservices into the Java EE ecosystem. MicroProfile is simply an open-source project led by Java EE vendors, is not a true open standard and thus can move much faster. MicroProfile recently had a 1.3 release and already enables features like fat-jars, dynamic configuration, circuit-breakers, fault-tolerance, health-checks, metrics, JWT, OpenAPI/Swagger, distributed tracing and type-safe REST clients. The Eclipse Foundation also has a project named JNoSQL that aims to standardize NoSQL database access for Java, much like JDBC. All of these features are likely to be standardized as part of the next major revision of Java EE.

In addition to announcing the release of Java EE 8, Oracle also announced that it is donating all Java EE code to the Eclipse Foundation. Going forward, Java EE will be renamed to Jakarta EE and will no longer be owned by any single commercial entity. It is expected that Jakarta EE 8 will be released later in the year with a fully open-source version of Java EE, including a completely open-source set of certification tests as well as an open-source governance process to replace the JCP. Jakarta EE 9 — including significant new features — is expected to be released in 2019.

REZA RAHMAN is Senior Vice President at AxonIQ. He has been an official Java technologist at Oracle. He is the author of the popular book EJB 3 in Action. Reza has long been a frequent speaker at Java User Groups and conferences worldwide including JavaOne and Devoxx. He has been the lead for the Java EE track at JavaOne as well as a JavaOne Rock Star Speaker award recipient. Reza is an avid contributor to industry journals like DZone and TheServerSide. He has been a member of the Java EE, EJB, and JMS expert groups over the years.

JAVA IS IN TENTS

With Oracle recently agreeing to release a new version of Java every six months, things in the Java world are about to be more intense than ever. Despite all the hype around Java 10, 64% of DZone users are still not using Java 9 or later, meaning that Java 8 remains very relevant. And as Oracle continues with new releases, it will be increasingly interesting to look back and see what has changed from version to version. Let's start doing that now by taking a hike to Camp Coffee to analyze the features of Java 8, 9, and 10 in tents.

JAVA 8

- Saw the introduction of many features related to functional programming into the Java language, such as lambda expressions, functional interfaces, and the Stream API.

- Introduced the Nashorn engine to enable running dynamic JavaScript code natively on the JVM.

- Included an overhaul for JavaFX, such as a new theme (Modena) and making it available for ARM platforms.

JAVA 9

- Modularized the Java Virtual Machine with Project Jigsaw (officially known as the Java Platform Module System).

- Brought a REPL to Java in the form of JShell — an interactive command-line interface.

- Improved and extended the JAR file format to allow for multiple Java release-specific versions of class files to

JAVA 10

- The first release in a new six-month cycle, Java 10 has been a relatively minor release leading up to Java 11.

- Brought local variable type inference to Java with the var keyword.

- Enabled Java's JIT compiler, Graal, to be used as an experimental compiler on the Linux/x64 platform.

ATTITUDES

Three quarters of the survey respondents feel optimistic about the future of Java, with 43% feeling fairly optimistic and 33% very optimistic.

JAKARTA EE

Formerly known as Java EE, focuses on distributed computing and web services.

SPRING FRAMEWORKS

is an open-source tool that provides infrastructure support for Java applications.

GARBAGE COLLECTION

A cleanup program and method of managing memory that runs on the JVM and removes objects that are not being used.

* *Features of all Java Versions*

Join the World's Largest Developer Community

Download the latest software, tools, and developer templates

Get exclusive access to hands-on trainings and workshops

Grow your network with the Developer Champion and Oracle ACE Programs

Publish your technical articles—and get paid to share your expertise

ORACLE DEVELOPER COMMUNITY developer.oracle.com
Membership Is Free | Follow Us on Social:

@OracleDevs   **facebook.com/OracleDevs**

ORACLE®

Shaping the Future of Java...Faster

Over the past 22 years, Java has grown into a vibrant community of more than 12 million developers. Moving forward, Oracle and the Java community are working to ensure Java continues to be well-positioned for modern application development and growth in the cloud.

In 2017, Oracle and the Java community announced its intentions to shift to a new six-month cadence for Java meant to reduce the latency between feature releases. At the same time, Oracle announced its plans to build and ship OpenJDK binaries. This release model takes inspiration from the release models used by other platforms and by various operating-system distributions addressing the modern application development landscape. Modern application development expects simple open licensing and a predictable time-based cadence, and the new release model delivers on both.

The first step on this journey was the release of Java SE 9 in September 2017. With over 100 enhancements, the defining feature of Java SE 9 was the Java Platform Module System, which makes it easier for developers to reliably assemble and maintain sophisticated applications. The module system also makes the JDK itself more flexible, allowing developers to bundle just those parts of the JDK that are needed to run an application when deploying to the cloud.

Oracle and the Java community are working to ensure Java continues to be well-positioned for modern application development and growth in the cloud.

Six months later in March 2018, Oracle released the general availability of Java SE 10, the first feature-based release as part of the new release cycle. The release is more than a simple stability and performance fix over Java SE 9; rather, it introduced twelve new enhancements defined through the JDK Enhancement Proposals that developers can immediate pick up and start using.

We're now preparing for Java SE 11, which is scheduled for general availability in September 2018 and currently targets eight enhancements, including JEP 309 (Dynamic Class-File Constants), JEP 323 (Local-Variable Syntax for Lambda Parameters), and JEP 328 (Flight Recorder).

Beyond Java SE 11, we're also investing in the next set of opportunities that Java can address such as containers, scalability, data optimization, hardware acceleration, and continued language enhancements:

- Portola: Java's characteristics make it ideal for container deployment, such as Docker, and the project goal to provide a port of the JDK to the Alpine Linux distribution.

- ZGC: The project goal is to provide a scalable, low latency garbage collector that does not require tuning.

- Valhalla: Java is very good at optimizing code and the project goal is to also optimize data.

- Panama: With opportunities in big data and machine learning, the project goal is to allow access to low-level hardware functionality through normal Java code.

- Amber: Coding productivity is key, and the project goal is to make Java a much less verbose and approachable language for developers.

The Java ecosystem continues to be a diverse collection of developers and we encourage them to join the OpenJDK Project to help shape the future of Java faster.

PRODUCT
Java SE

BLOG
blogs.oracle.com/
java-platform-group

TWITTER
@OpenJDK

WEBSITE
openjdk.java.net

ORACLE®

BY SHARAT CHANDER, DIRECTOR, JAVA PLATFORM PRODUCT MANAGEMENT AND DEVELOPER RELATIONS

Free, Fast, Open, Production-Proven, and All Java: OpenJ9

QUICK VIEW

01. Eclipse OpenJ9 is optimized to run Java applications cost-effectively in the cloud.

02. OpenJ9 can replace HotSpot in OpenJDK builds and can be used with a wide range of system architectures and operating systems.

03. The AdoptOpenJDK project releases pre-built binaries for Linux x64, Linux s390x, Linux ppc64, and AIX ppc64 for you to get started.

BY MARKUS EISELE
DIRECTOR OF DEVELOPER ADVOCACY, **LIGHTBEND**

Eclipse OpenJ9 is a high-performance, scalable, Java virtual machine (JVM) implementation. It is not new. It has been part of the IBM Java Development Kit for many years and it is safe to say that it is an enterprise-grade, production-level component of many large Java-based systems. At the end of last year, it was contributed to the Eclipse Foundation by IBM. OpenJ9 is an alternative to the Hotspot JVM currently mostly used within OpenJDK.

THE HISTORY

Although the Eclipse OpenJ9 project hasn't been around for very long, the VM itself has been around for years. It was launched during the fourth JavaOne and parts of the codebase can be traced back to Smalltalk. It has been powering IBM middleware products for the last decade or more. IBM contributed the VM to the Eclipse Foundation end of 2017 and more than 70 IBM developers are actively involved in the project.

The long-term goal of the Eclipse OpenJ9 project is to foster an open ecosystem of JVM developers that can collaborate and innovate with designers and developers of hardware platforms, operating systems, tools, and frameworks. The Java community has benefited over its history from having multiple implementations of the JVM specification competing to provide the best runtime for your application. Whether adding compressed references, new cloud features, AOT (ahead-of-time) compilation, or straight up faster performance and lower memory use, the ecosystem has improved through that competition. Eclipse OpenJ9 aims to continue to spur innovation in the runtimes space.

As an essential part of this, the teams work closely with the AdoptOpenJDK efforts, which are led by the London Java Community. If you look closely, you realize that a large part of OpenJ9 has been living at the Eclipse Foundation for a couple of years now. Eclipse OpenJ9 embeds Eclipse OMR, which provides cross platform components for building reliable, high-performance language runtimes. Eclipse OpenJ9 takes OMR and adds extra code that turns it into a runtime environment for Java applications.

WHAT EXACTLY IS OPENJ9 AND HOW DOES IT RELATE TO OPENJDK?

OpenJ9 replaces HotSpot JVM in the OpenJDK build and becomes a new JVM runtime implementation that can be used with a wide range of system architectures and operating systems. Currently, there are pre-built OpenJDK binaries with OpenJ9 as the JVM available for Java versions 8, 9, and 10 via the AdopOpenJDK project website. A JVM is not the complete Java Development Kit (JDK). It is the execution engine. While the JDK provides tools, class libraries, and more general things that you need to build your application, the JVM is responsible to run the actual bytecode. It is a replacement for the commonly used and known Hotspot JVM which was implemented by Oracle.

In order to create an OpenJDK build with OpenJ9 as the JVM, the OpenJDK project is mirrored into a separate repository that removes Hotspot and provides a place for the needed integration code. The integration code consists mostly of changes to Makefiles and configuration, some patches to the class library code (JCL), and some minor patches for additional OpenJ9 features. The integration layer mitigates the few places where the interfaces from Hotspot and OpenJ9 are slightly different. All in all, it is roughly 4,000 lines of code, which is really small compared to the size of OpenJ9 or the OpenJDK. The primary goal is to keep this to a necessary minimum and also contribute as much as possible upstream to the OpenJDK.
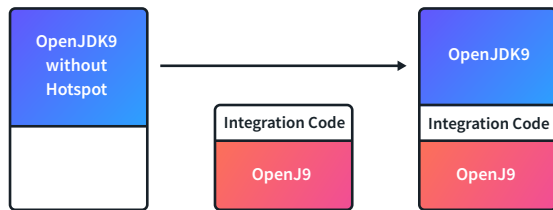
With OpenJDK, with the integration code and OpenJ9 coming together via a build process, the end result is a binary that is available for a range of different platforms. You can find more information about the build process and how to build your own OpenJDK with OpenJ9 on the project website.

WHY OPENJ9 IS THE PERFECT CLOUD RUNTIME

The requirements of cloud runtimes today are different to the characteristics employed by individual, physical machines in the datacenters from just a couple of years ago. A small memory footprint allows a higher application density for providers and reduces the cost per application for users as they can run more instances with less resources. Another important feature is a fast startup, which lets application instances scale faster and more smoothly.

The OpenJ9 project uses the Daytrader application to monitor performance improvements in comparison with Hotspot. With a 66% smaller memory footprint after startup and a 42% faster startup time, it reaches a comparable throughput in a constraint environment like a Docker container.

UNIQUE FEATURES AND ADVANTAGES

The lower memory footprint and faster startup times demonstrate the practical impact of some of the advantages and implementation differences over the classical Hotspot JVM.

APPLICATION CLASS SHARING AND AOT (AHEAD-OF-TIME) COMPILATION

The first time an application runs, the JVM has to load all the classes that are required to start the application. This process takes time. If the needed classes are stored in cache, the second application run will be much faster. Another implementation difference is that the JVM doesn't store the Java bytecode classes but an optimized ROMClass version of it. These read-only, optimized versions are carrying symbolic data only. When the JVM is executing a class, it has to be able to do a bunch of lookups, e.g. finding the method it is actually invoking, and it also needs to be able to work on data and save this data. This is where the J9RAMClass comes in. It is a cache for the data at runtime and carries the live data for a particular class.

Further on, the ROMClasses can be shared between multiple JVMs on the same machine. OpenJ9 always shares both the bootstrap and application classes that are loaded by the default system class loader. A detailed view on how this works is available in this article. And there is something else that can be optimized with this shared class cache ahead-of-time compilation (AOT). Unlike just-in-time (JIT) compilation, AOT is dynamically compiling methods into ATO code at

runtime and placing these ROMClasses into the shared cache. The VM automatically chooses which methods should be AOT-compiled based on heuristics that identify the start-up phase of large applications. It is enabled via the `-Xshareclasses` option, which is highly configurable. As for AOT itself, it works straight out of the box when you enable shared classes and doesn't require any special tuning.

TUNING FOR CLOUD ENVIRONMENTS

The `-Xtune:virtualized` option is designed to configure OpenJ9 for typical cloud deployments where VM guests are provisioned with a small number of virtual CPUs to maximize the number of applications that can be run. The `-Xquickstart` option enables an ultra-fast startup of OpenJ9 and works best with short-lived tasks. But be aware that you may trade in the peak throughput capabilities. You can also specify `-XX:+IdleTuningGcOnIdle` on the command line. When set, OpenJ9 determines whether an application is idle based on CPU utilization and other internal heuristics. When an idle state is recognized, a GC cycle runs if there is significant garbage in the heap and releases unused memory back to the operating system. A more detailed overview is provided in this article.

GARBAGE COLLECTION IN OPENJ9

Eclipse OpenJ9 has a number of GC policies designed around different types of applications and workloads. The Generational Concurrent (`-Xgcpolicy:gencon`) GC policy is the default policy employed by the JVM. But there are four other alternatives: `-Xgcpolicy:balanced`, `-Xgcpolicy:metronome`, `-Xgcpolicy:optavgpause`, and `-Xgcpolicy:optthruput`. The Metronome GC (`-Xgcpolicy:metronome`) policy is especially interesting if your application depends on precise response times and you are running on x86 Linux. The main difference between Metronome and other policies is that garbage collection occurs in small interruptible steps rather than stopping an application completely while garbage is marked and collected. By default, Metronome pauses for three milliseconds at a time. A full garbage collection cycle requires many pauses, which are spread out to give the application enough time to run. You can limit the amount of CPU that the GC process uses and you can control the pause time. A more complete overview about the individual policies can be found in this article.

GET STARTED

The AdoptOpenJDK project releases pre-built binaries for Linux x64, Linux s390x, Linux ppc64, and AIX ppc64. The easiest way to get started is to download the one for your operating system and start working with it — although you have the ability to build binaries for other platforms like Linux ARMv7 yourself. Docker images are available via DockerHub.

The JVM documentation is extensive and well-structured to get you started with optimizing for your application.

MARKUS EISELE is a Java Champion, former Java EE Expert Group member, founder of JavaLand, reputed speaker at Java conferences around the world, and a very well known figure in the Enterprise Java world. He works as Director of Developer Advocacy for Lightbend, Inc. Follow his latest updates and Java news @myfear on Twitter.

2018 Java Ecosystem Executive Insights

BY TOM SMITH
RESEARCH ANALYST AT **DZONE**

QUICK VIEW

01. Java continues to be the platform of choice for a number of enterprise companies as a result of its portability and because it allows developers to write once run anywhere.

02. The JVM is the most critical element of the Java ecosystem followed by its openness, compatibility, the vastness of the libraries, and the completeness of the toolchains.

03. The most significant changes to the ecosystem in the past year have been the move to semi-annual releases and Java EE moving to the Eclipse Foundation as Jakarta EE.

To gather insights on the current and future state of the Java ecosystem, we talked to executives from 14 companies. Here's who we spoke to:

RESPONDENTS

- **Gil Tayar**, Senior Architect and Evangelist, **Applitools**

- **Frans van Buul**, Commercial Developer, Evangelist, **AxonIQ**

- **Carlos Sanches**, Software Engineer, **CloudBees**

- **Jeff Williams**, Co-founder and CTO, **Contrast Security**

- **Doug Pearson**, CTO, **FlowPlay**

- **John Duimovich**, Distinguished Engineer and Java CTO, **IBM**

- **Brian Pontarelli**, CEO, **Inversoft**

- **Wayne Citrin**, CTO, **JNBridge**

- **Ray Augé**, Sr. Software Architect, **Liferay**

- **Matt Raible**, Java Champion and Developer Advocate, **Okta**

- **Heather VanCura**, Chair of the Java Community Process Program, **Oracle**

- **Burr Sutter**, Director Developer Experience, **Red Hat**

- **Ola Petersson**, Software Consultant, **Squeed**

- **Roman Shoposhnik**, Co-founder, V.P. Product and Strategy, **Zededa**

KEY FINDINGS

While Java is still the prevalent language in enterprises and there are more Java developers than anything else, this topic does not generate the level of response of the other topics for which we produce research guides. Java content continues to outperform all other content on DZone by a 4:1 margin — but because it's not new and "sexy," not a lot of people want to talk about it. I'm most appreciative to the IT professionals that took the time to share their insights on the current and future state of the Java ecosystem.

1. Java continues to be the platform of choice for a number of enterprise companies, including financial institutions, as a result of its portability and because it allows developers to write once run anywhere. There are plenty of Java developers and it is seeing a resurgence with big data.

2. The Java Virtual Machine (JVM) was most frequently mentioned as the most important element of the Java ecosystem. The JVM is the most critical element followed by its openness, compatibility, the vastness of the libraries, and the completeness of the toolchains. The JVM enables languages other than Java to flourish. The fact that Java is open-source while championed by a large company was deemed to be important by a couple of respondents, as was the community in which no one participant is more important than the community.

The maturity level of Java is high. This results in a lot of frameworks, libraries, and IDEs as well as a high-performance, consistent, compatible language that's simple and stable.

3. The most important player in the Java ecosystem is Oracle followed by multiple mentions of IBM, the Apache and Eclipse foundations, Red Hat, and Pivotal. Oracle was seen as the key holder but seems to be pulling back.

It was interesting that financial institutions as a whole were mentioned given their use of Java and the number of developers

they employ along with other companies like Twitter, Alibaba, Facebook, and Google with GCP and Kubernetes.

The Eclipse Foundation is likely to become the most significant player with its efforts around MicroProfile and Jakarta EE.

4. The two most significant changes to the ecosystem in the past year have been the move to **semi-annual releases and Java EE moving to the Eclipse Foundation as Jakarta EE**, putting the future of enterprise Java into the hands of the community. The move to half-yearly releases will boost interest in, and use of, Java by developers.

The "open-souring" of Java EE to the Eclipse Foundation, creating EE4J and now the birth of Jakarta EE, is significant. Java has been the dominant player in enterprise applications for two decades. Jakarta EE ensures Java will continue to be the dominant player for enterprise computing for a long time.

5. The availability of developers helps Java solve many problems in organizations. There is a shortage of security, big data, and AI/ML/DL/NLP professionals, but there are plenty of Java developers who can get organizations' work done. Twitter, financial institutions, automobile manufacturers, and AI/ML companies are all heavy users of Java. Java has the ability to support high-speed concurrent processing at scale, which becomes more important as the amount of data continues to grow.

6. The people I spoke with are all Java devotees and **don't see any problems with Java**. They realize that those who are not Java developers see the language as being too verbose. They believe Eclipse is a good steward of open source and, like open source, there needs to be more participation and engagement in the community.

While Java is considered verbose by some, there are alternatives like Kotlin, Scala, and project Lombok. Java lags behind because it's heavily used by large enterprises. With slowness comes stability. While it lacks some of the niceties of other languages, it provides quick wins with fast coding,

7. Serverless was mentioned by a couple of respondents as the future of Java. They believe it will lead to a major reshaping. Java is built for serverless, but it needs work. With Spring Boot, containers can be lighter-weight and great to build serverless upon.

There were good changes in Java 8 and 9 for easier execution in container management, memory, and CPU. JVM-based languages and tooling will continue to evolve. The JVM enables many different types of languages to be built.

Others see Java thriving in the open-source software ecosystem with innovations continuing to support Java's ongoing success.

8. The biggest concerns with the current state of the Java ecosystem were **quality deterioration because people were not learning**

from their mistakes, understanding the value of the ecosystem, or taking security seriously enough.

Some question the benefits of the more frequent release cycles because it may lead to release fatigue and meaningless releases that ultimately will not be adopted or supported.

9. When working with Java, developers need to **realize the depth of the ecosystem and not try to reinvent the wheel**. Learn the libraries and know your code is going to be attacked — prepare accordingly. Pay attention to the Java roadmap and try new builds before they go live to be seen as an expert. Like any other language, ensure your code is well-designed, extendable, maintainable, and easily understood by others.

JVM is the top-performing platform. Most languages live on the JVM so start with Java. Be language-agnostic. Learn domain-driven design. Read *Design Patterns* by the Gang of Four. There's a good future with Java. Look for tools to help with development. Keep an eye on open-source projects through good information outlets.

10. Additional considerations regarding Java include:

a. Containers are changing how developers deploy applications and that affects Java applications.

b. Pay attention to Kotlin which runs on the JVM. Adopt new languages to help create new applications. There are a lot of really smart people in the ecosystem; pay attention to what they have to say. There's a lot to continue learning since things will continue to change.

c. Java developers' participation matters in the continued consistency, stability, and security of the Java ecosystem.

d. One of the strengths of Java that's undervalued is that it's a small language that does not offer a huge range of options for how to do things. As a result, two engineers writing the same solution for the same problem tend to end up with the same code. This is a strong advantage of Java that is not true of many other competing languages.

e. We need to recognize the importance of open source — embrace it and contribute to it where we can. If you find problems, work on fixing them. It's a great community and it's made better when everyone is involved and contributing.

TOM SMITH is a Research Analyst at DZone who excels at gathering insights from analytics—both quantitative and qualitative—to drive business results. His passion is sharing information of value to help people succeed. In his spare time, you can find him either eating at Chipotle or working out at the gym.

Solutions Directory

Java gets even greater when you have the right tools to back you up. This directory contains libraries, frameworks, IDEs, and more to help you with everything from database connection to release automation, from code review to application monitoring, from microservice architectures to memory management. Amp up your Java development with these solutions to make your life easier and your application more powerful.

COMPANY	PRODUCT	PRODUCT TYPE	VERTICAL	WEBSITE
Adzerk	Hoplon	ClojureScript web framework	Open source	hoplon.io
Alachisoft	TayzGrid	In-memory data grid (JCache-compliant)	Open source	alachisoft.com/tayzgrid
Amazon Web Services	AWS ECS	Elastic container service w/Docker support	Free tier available	aws.amazon.com/ecs
AngularFaces	AngularFaces	AngularJS plus JSF	Open source	angularfaces.com
ANTLR	ANTLR v3	Parser generator for creating compilers & related tools	Open source	antlr3.org
Apache Software Foundation	Apache Ant	Build automation (process-agnostic: specify targets & tasks)	Open source	ant.apache.org
Apache Software Foundation	Apache Camel	Java implementation of enteprise integration patterns	Open source	camel.apache.org
Apache Software Foundation	Apache Commons	Massive Java package collection	Open source	commons.apache.org/components
Apache Software Foundation	Apache Commons DBCP	Database connection pooling	Open source	commons.apache.org/proper/commons-dbcp
Apache Software Foundation	Apache Commons IO	Utilities for Java I/O	Open source	commons.apache.org/proper/commons-io
Apache Software Foundation	Apache CXF	Java services framework with JAX-WS & JAX-RS support	Open source	cxf.apache.org

COMPANY	PRODUCT	PRODUCT TYPE	VERTICAL	WEBSITE
Apache Software Foundation	Apache DeltaSpike	Portable CDI extensions (bean validation, JSF enhancements, invocation controls, transactions contexts)	Open source	deltaspike.apache.org
Apache Software Foundation	Apache Ignite	In-memory data grid	Open source	ignite.apache.org
Apache Software Foundation	Apache Ivy	Dependency management with strong Ant integration	Open source	ant.apache.org/ivy
Apache Software Foundation	Apache Kafka	Distributed pub-sub message broker	Open source	kafka.apache.org
Apache Software Foundation	Apache Log4j 2	Logging for Java	Open source	logging.apache.org/log4j/2.x
Apache Software Foundation	Apache Lucene	Search engine in Java	Open source	lucene.apache.org/core
Apache Software Foundation	Apache Maven	Build automation (opinionated, plugin-happy, higher-level build phases, dependency management/resolution)	Open source	maven.apache.org
Apache Software Foundation	Apache Mesos	Distributed systems kernel	Open source	mesos.apache.org
Apache Software Foundation	Apache MyFaces	JSF plus additional UI widgets, extensions, & integrations	Open source	myfaces.apache.org
Apache Software Foundation	Apache OpenNLP	Natural language processing machine learning toolkit	Open source	opennlp.apache.org
Apache Software Foundation	Apache POI	Microsoft document processing for Java	Open source	poi.apache.org
Apache Software Foundation	Apache Shiro	Java security framework (authen/author, crypto, session management)	Open source	shiro.apache.org
Apache Software Foundation	Apache Struts	Web framework (servlet & MVC)	Open source	struts.apache.org
Apache Software Foundation	Apache Tapestry	Web framework (POJOs, live class reloading, opinionated, light HttpSessions)	Open source	tapestry.apache.org
Apache Software Foundation	Apache Tomcat	Servlet container & web server (JSP, EL, Websocket)	Open source	tomcat.apache.org

COMPANY	PRODUCT	PRODUCT TYPE	VERTICAL	WEBSITE
Apache Software Foundation	Apache Wicket	Simple web app framework (pure Java & HTML w/Ajax output)	Open source	wicket.apache.org
Apache Software Foundation	Apache Xerces2	XML parser for Java	Open source	xerces.apache.org/xerces2-j
Apache Software Foundation	Derby	Java SQL database engine	Open source	db.apache.org/derby
Apache Software Foundation	FreeMarker	Server-side Java web templating (static & dynamic)	Open source	freemarker.apache.org
Apache Software Foundation	Apache TomEE	Apache Tomcat & Java EE features (CDI, EJB, JPA, JSF, JSP)	Open source	tomee.apache.org
AppDynamics	AppDynamics	APM w/Java agent	15 days	appdynamics.com
AssertJ	AssertJ	Java assertion framework (for verification & debugging)	Open source	joel-costigliola.github.io/assertj
Atlassian	Clover	Code coverage analysis tool	Open source	atlassian.com/software/clover
Azul Systems	jHiccup	Show performance issues caused by JVM (as opposed to app code)	Open source	azul.com/jhiccup
Azul Systems	Zing	JVM w/unique pauseless GC	Free tier available	azul.com/products/zing
Azul Systems	Zulu	Enterprise-grade OpenJDK build	Open source	azul.com/products/zulu
Synopsys	Black Duck Platform	Security & open-source scanning & management (w/container support)	Open source	blackducksoftware.com
BMC	TrueSight Pulse	Infrastructure monitoring	14 days	truesightpulse.bmc.com
Bouncy Castle	Bouncy Castle	Java & C# cryptography libraries	Open source	bouncycastle.org
CA Technologies	CA Application Monitoring	APM w/Java agent	30 days	ca.com/us/products/ca-application-performance-management.html
Canoo	Dolphin Platform	Presentation model framework (multiple views for the same MVC group)	Open source	github.com/canoo/dolphin-platform

COMPANY	PRODUCT	PRODUCT TYPE	VERTICAL	WEBSITE
Cask	Cask	Data & application integration platform	Open source	cask.co
Catchpoint	Catchpoint	APM w/Java agent	14 days	catchpoint.com
Charlie Hubbard	Flexjson	JSON serialization	Open source	flexjson.sourceforge.net
CheckStyle	CheckStyle	Automated check against Java coding standards	Open source	checkstyle.sourceforge.net
Chef Software	Chef	Infrastructure automation/ configuration management	Open source	chef.io/chef
Chronon Systems	DripStat APM	Java & Scala APM w/many framework integrations	Free tier available	dripstat.com/products/apm
Cloudbees	Cloudbees Jenkins Platform	CI server & verified plugins, build server provisioning, pipeline monitoring, build analytics	14 days	cloudbees.com
Cloudbees	Jenkins	CI server	Open source	jenkins.io
Codeborne	Selenide	UI tests in Java (Selenium WebDriver)	Open source	selenide.org
Red Hat	Codenvy IDE	SaaS IDE w/dev workspace isolation	Free tier available	codenvy.com
Couchbase	Couchbase	Document-oriented DBMS	Open source	couchbase.com
Cucumber	Cucumber	BDD framework w/Java version	Open source	cucumber.io
Data Geekery	jOOQ	Non-ORM SQL in Java	Open source version available	jooq.org
Data Geekery	jOOλ	Extension of Java 8 lambda support (tuples, more parameters, sequential & ordered streams)	Open source	github.com/jOOQ/jOOL
Docker	Docker	Containerization platform	Open source	docker.com
Draios	Sysdig	Container monitoring	Open source	sysdig.com

COMPANY	PRODUCT	PRODUCT TYPE	VERTICAL	WEBSITE
Dynatrace	Dynatrace Application Monitoring	APM	15 days	dynatrace.com
EasyMock	EasyMock	Unit testing framework (mocks Java objects)	Open source	easymock.org
Eclipse Foundation	Eclipse	IDE (plugin-happy)	Open source	eclipse.org
Eclipse Foundation	Eclipse Che	IDE (workspace isolation, cloud hosting)	Open source	eclipse.org/che
Eclipse Foundation	Eclipse Collections	Java Collections framework	Open source	eclipse.org/collections
Eclipse Foundation	EclipseLink	JPA & MOXx (JAXB) implementation	Open source	eclipse.org/eclipselink
Eclipse Foundation	Jetty	Servlet engine & HTTP server (w/non-HTTP protocols)	Open source	eclipse.org/jetty
Eclipse Foundation	SWT	Java UI widget toolkit	Open source	eclipse.org/swt
EJ Technologies	JProfiler	Java profiling	Free for open source and nonprofits	ej-technologies.com/products/jprofiler/overview.html
Elastic	ElasticSearch	Distributed search & analytics engine	Open source	elastic.co
Electric Cloud	ElectricFlow	Release automation	Free version available	electric-cloud.com
Elide	Elide	JSON <- JPA web service library	Open source	elide.io
GE Software	Predix	Industrial IoT platform w/Java SDK (on Cloud Foundry)	N/A	ge.com/digital/predix
Genuitec	MyEclipse	IDE (Java EE & web)	30 days	genuitec.com/products/myeclipse
Google	Google Web Toolkit (GWT)	Java -> Ajax	Open source	gwtproject.org
Google	GSON	JSON serialization	Open source	github.com/google/gson

COMPANY	PRODUCT	PRODUCT TYPE	VERTICAL	WEBSITE
Google	Guava	Java libraries from Google (collections, caching, concurrency, annotations, I/O)	Open source	github.com/google/guava
Google	Guice	Dependency injection framework	Open source	github.com/google/guice
Gradle	Gradle	Build automation (Groovy-based scripting of task DAGs)	Open source	gradle.org
GridGain Systems	GridGain	In-memory data grid (Apache Ignite & enterprise management, security, monitoring)	30 days	gridgain.com
H2	H2	Java SQL database engine	Open source	h2database.com
Haulmont	CUBA Platform	Java rapid enterprise app development framework	Free tier available	cuba-platform.com
Hazelcast	Hazelcast Enterprise Platform	Distibuted in-memory data grid (w/ JCache implementation)	30 days	hazelcast.com
HyperGrid	HyperForm	Container composition platform	Available by request	hypergrid.com
IBM	IBM Cloud	PaaS w/extensive Java support	Free tier available	ibm.com/cloud
IBM	WebSphere Application Server	Java application server	Available by request	ibm.com/cloud/websphere-application-platform
IBM	WebSphere eXtreme Scale	In-memory data grid	Available by request	ibm.com/support/knowledgecenter/en/SSTVLU_8.5.0/com.ibm.websphere.extremescale.doc/cxsoverview.html
IceSoft	IceFaces	JSF framework	Open source	icesoft.org/java/projects/ICEfaces/overview.jsf
Immunio	Immunio	Runtime application self-protection w/Java support	Free tier available	immun.io
Informatica	Informatica	Data integration & management	30 days	informatica.com
Integral	FusionReactor	JVM APM w/production debugging and crash protection	14 days	fusion-reactor.com

COMPANY	PRODUCT	PRODUCT TYPE	VERTICAL	WEBSITE
Isomorphic Software	SmartGWT	Java -> Ajax w/rapid dev tools, UI components, multi-device	60 days	smartclient.com/product/smartgwt.jsp
iText Group	iText 7	PDF manipulation from Java	30 days	itextpdf.com
Jackson	Jackson	JSON processing	Open Source	github.com/FasterXML/jackson
Jahia Solutions Group	Jahia Platform	Enterprise CMS/portal (Jackrabbit compliant)	Open-source version available	jahia.com
Janino Compiler	JANINO	Lightweight Java compiler	Open source	janino-compiler.github.io/janino
jClarity	Censum	GC log analysis	7 days	jclarity.com/censum
jClarity	Illuminate	Java-focused APM w/machine learning & autosummarization	14 days	jclarity.com/illuminate
Java Decompiler	JD	Java decompiler	Open source	jd.benow.ca
Jdbi	Jdbi	SQL library for Java	Open source	jdbi.org
JDOM	JDOM	XML in Java (w/DOM & SAX integration)	Open source	jdom.org
Jelastic	Jelastic	Multi-cloud PaaS (w/Java support)	Available by request	jelastic.com
JetBrains	Upsource	Code review	Free 10-user plan	jetbrains.com/upsource
JetBrains	IntelliJ IDEA	IDE	Free tier available	jetbrains.com/idea
JFrog	Artifactory	Package hosting & distribution infrastructure	Available by request	jfrog.com/artifactory
JFrog	Bintray	Package hosting & distribution infrastructure	Available by request	bintray.com
Jinfonet	JReport	Reporting, dashboard, analytics, & BI for Java	Available by request	jinfonet.com

COMPANY	PRODUCT	PRODUCT TYPE	VERTICAL	WEBSITE
JNBridge	JMS Adapters for .NET or BizTalk by JNBridge	JMS Integration & .NET or BizTalk	30 Days	jnbridge.com/software/jms-adapter-for-biztalk/overview
JNBridge	JNBridgePro	Java & .NET interoperability	30 Days	jnbridge.com/software/jnbridgepro/overview
Joda	Joda-Time	Low-level Java libraries	Open source	joda.org/joda-time
Joyent	Triton	Container-native infrastructure w/ Java images	$250 credit	joyent.com/triton/compute
JUnit	JUnit 5	Unit testing framework (mocks Java objects)	Open source	junit.org/junit5
Liferay	Liferay Digital Experience Platform	Enterprise CMS/portal	Open source version available	liferay.com
Lightbend	Akka	Java implementation of Actor Model	Open source	akka.io
Lightbend	Lagom	Reactive microservices framework (Java, Scala)	Open source	lightbend.com/lagom-framework
Lightbend	Lightbend Reactive Platform	Dev & prod suite for reactive JVM applications (Akka, Play, Lagom, Spark)	Open source	lightbend.com/products/reactive-platform
Lightbend	Play	Java & Scala web framework (stateless, async, built on Akka)	Open source	playframework.com
Lightbend	Spray	REST for Scala/Akka	Open source	spray.io
Machinery for Change	CP3O	JDBC connection & statement pooling	Open source	mchange.com/projects/c3p0
MarkLogic	MarkLogic 8	Multi-model enterprise NoSQLÊdatabase	Free tier available	marklogic.com
Mendix	Mendix Platform	Enterprise aPaaS	Available by request	mendix.com/application-platform-as-a-service
Microfocus	Visual COBOL	COBOL accessibility from Java (w/COBOL -> Java bytecode compilation)	30 days	microfocus.com/products/visual-cobol

COMPANY	PRODUCT	PRODUCT TYPE	VERTICAL	WEBSITE
Mockito	Mockito	Unit testing framework (mocks Java objects)	Open source	mockito.org
MongoDB	MongoDB	Document-oriented DBMS	Open source	mongodb.com
Mozilla	Rhino	JavaScript implemention in Java (for embedded JS)	Open source	developer.mozilla.org/en-US/docs/Mozilla/Projects/Rhino
MuleSoft	AnyPoint Platform	Hybrid integration platform	Available by request	mulesoft.com/platform/enterprise-integration
MyBatis	MyBatis	JDBC persistence framework	Open source	mybatis.org/mybatis-3
Mysema	Querydsl	DSL for multiple query targets (JPA, JDO, SQL, Lucene, MongoDB, Java Collections)	Open source	querydsl.com
Nastel	AutoPilot	APM	Demo available by request	nastel.com
Netflix	Hystrix	Latency and fault tolerance library	Open source	github.com/Netflix/Hystrix
Netflix	Ribbon	RPC library w/load balancing	Open source	github.com/Netflix/ribbon
Netflix	RxJava	Reactive extension for JVM (extends observer pattern)	Open source	github.com/ReactiveX/RxJava
New Relic	New Relic	APM w/Java agent	Demo available by request	newrelic.com
NGINX	NGINX	Web server, load balancer, reverse proxy	Open source	nginx.com
Ninja Framework	Ninja Framework	Full-stack web framework for Java	Open source	ninjaframework.org
Nuxeo	Nuxeo Platform	Structured & rich content management platform	Demo available by request	nuxeo.com
Object Refinery Limited	JFreeChart	Java charting library	Open source	jfree.org/jfreechart
OmniFaces	OmniFaces	JSF utility library	Open source	omnifaces.org

COMPANY	PRODUCT	PRODUCT TYPE	VERTICAL	WEBSITE
OpenCV Team	OpenCV	Computer vision libraries (w/Java interfaces)	Open source	opencv.org
Oracle	GlassFish 5	Java application server	Open source	javaee.github.io/glassfish/download
Oracle	JavaFX	Java GUI library	Open source	docs.oracle.com/javase/8/javase-clienttechnologies.htm
Oracle	JAX-RS	REST spec for Java	Open source	download.oracle.com/otndocs/jcp/jaxrs-2_0-fr-eval-spec
Oracle	JDeveloper	IDE	Open source	oracle.com/technetwork/developer-tools/jdev/overview/index-094652.html
Oracle	Jersey	RESTful web services in Java (JAX-RS w/enhancements)	Open source	github.com/jersey/jersey
Oracle	JavaServer Faces	Java spec for server-side component-based UI	Open source	oracle.com/technetwork/java/javaee/javaserverfaces-139869.html
Oracle	JSP	Server-side Java web templating (static & dynamic)	Open source	oracle.com/technetwork/java/javaee/jsp
Oracle	NetBeans	IDE	Open source	netbeans.org
Oracle	Oracle Coherence	In-memory ditributed data grid	Open source	oracle.com/technetwork/middleware/coherence/overview
Oracle	Oracle Database 18c	Relational DBMS	N/A	oracle.com/technetwork/database/index.html
Oracle	VisualVM	JVM monitoring	Open source	visualvm.github.io
Oracle	WebLogic	Java application server	N/A	oracle.com/middleware/weblogic
OSGi Alliance	OSGi	Dynamic component system spec for Java	Open source	osgi.org
OutSystems	OutSystems	Rapid application development platform	Available by request	outsystems.com
OverOps	OverOps	JVM agent for production debugging	Available by request	overops.com

COMPANY	PRODUCT	PRODUCT TYPE	VERTICAL	WEBSITE
OW2 Consortium	ASM	Java bytecode manipulation & analysis framework	Open source	asm.ow2.io
Payara	Payara Server	Java EE application server (enchanced GlassFish)	Open source	payara.fish/home
Pedestal	Pedestal	Clojure web framework	Open source	github.com/pedestal/pedestal
Percona	Percona Server	High-performance drop-in MySQL or MongoDB replacement	Open source	percona.com
Pivotal	GemFire	Distributed in-memory data grid (using Apache Geode)	Open source	pivotal.io/big-data/pivotal-gemfire
Pivotal & Spring	Project Reactor	Non-blocking, async JVM library (based on Reactive Streams spec)	Open source	projectreactor.io
Pivotal	Spring Boot	REST web services framework (opinionated, rapid spinup)	Open source	projects.spring.io/spring-boot
Pivotal	Spring Cloud	Distributed systems framework (declarative, opinionated)	Open source	cloud.spring.io
Pivotal	Spring Framework	Enterpise Java platform (large family of convention-over-configuration services, e.g. dependency injection, MVC, messaging, testing, AOP, data access, distributed computing services)	Open source	projects.spring.io/spring-framework
Pivotal	Spring MVC	Server-side web framework	Open source	docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html
Plumbr	Plumbr	Memory leak detection, GC analysis, thread & query monitoring	Demo available by request	plumbr.io
PrimeTek	PrimeFaces	UI components for JSF	Open source	primefaces.org
Progress Software	DataDirect	JDBC connectors (many data sources)	Available by request	progress.com/jdbc
PTC	ThingWorx	IoT platform w/Java SDK		developer.thingworx.com
PubNub	PubNub	Real-time mobile, web, & IoT APIs	Free tier available	pubnub.com

COMPANY	PRODUCT	PRODUCT TYPE	VERTICAL	WEBSITE
Puppet Labs	Puppet	Infrastructure automation/ configuration management	Open source	puppet.com
Push Technology	Diffusion	Real-time messaging (web, mobile, IoT)	Available by request	pushtechnology.com
Qoppa Software	Qoppa PDF Studio	PDF manipulation from Java	Available by request	qoppa.com
QOS.ch	Logback	Java logging framework	Open source	logback.qos.ch
QOS.ch	Sl4j	Logging for Java	Open source	slf4j.org
Raphael Winterhalter	CGLIB	Byte code generation library	Open source	github.com/cglib/cglib
Red Hat	Ansible	Deployment automation & configuration management	Open source	ansible.com
Red Hat	Drools	Business rules management system	Open source	drools.org
Red Hat	Hibernate ORM	Java ORM w/JPA & native APIs	Open source	hibernate.org/orm
Red Hat	Hibernate Search	Full-text search for objects (indexes domain model w/annotations, returns objects from free text queries)	Open source	hibernate.org/search
Red Hat	Infinispan	Distributed in-memory key/value store (Java-embeddable)	Open source	infinispan.org
Red Hat	JBoss Data Grid	In-memory distributed NoSQL data store	Available by request	redhat.com/en/technologies/ jboss-middleware/data-grid
Red Hat	JBoss EAP	Java EE 7 platform	Open source	developers.redhat.com/ products/eap/overview
Red Hat	JGroups	Java multicast messaging library	Open source	jgroups.org
Red Hat	RichFaces	UI components for JSF	Open source	richfaces.jboss.org

COMPANY	PRODUCT	PRODUCT TYPE	VERTICAL	WEBSITE
Red Hat	WildFly	Java application server	Open source	wildfly.org
Red Hat	WildFly Swarm	Uber JAR builder (w/trimmed WildFly app server)	Open source	wildfly-swarm.io
Redis Labs	Redis	In-memory key-value data structure store (use as DB, cache, message broker)	Open source	redis.io
Ring	Ring	Clojure web framework	Open source	github.com/ring-clojure/ring
Riverbed	SteelCentral	APM	10-90 days	riverbed.com
Salesforce	Heroku Platform	PaaS	Free tier available	heroku.com
Salesforce	Salesforce App Cloud	PaaS w/app marketplace	Free developer version	developer.salesforce.com
Sauce Labs	Sauce Labs Automated Testing Platform	Browser & mobile test automation (Selenium, Appium) w/Java interface	Open source	saucelabs.com/open-source
Scalatra Team	Scalatra	Scala web microframework	Open source	scalatra.org
Slenium	Selenium	Browser automation w/Junit and TestNG integration	Open source	seleniumhq.org
SonarSource	SonarQube	Software quality platform (unit testing, code metrics, architecture & complexity analysis, coding rule checks)	Open source	sonarqube.org
Sonatype	Nexus Repository	Binary/artifact repository	Available by request	sonatype.com/nexus-repository-sonatype
Spark	Spark Framework	Lightweight Java 8 web app framework	Open source	sparkjava.com
Spock	Spock	Test and specification framework for Java & Groovy	Open source	spockframework.org
Square	Dagger	Dependency injector for Android & Java	Open source	github.com/google/dagger

COMPANY	PRODUCT	PRODUCT TYPE	VERTICAL	WEBSITE
Teradata	Teradata	Data warehousing, analytics, lake, SQL on Hadoop and Cassandra, big data appliances, R integration, workload management	N/A	teradata.com
Terracotta	BigMemory Max	In-memory data grid w/Ehcache (JCache implementation)	Available by request	terracotta.org/bigmemory-and-web-sessions
Terracotta	EHCache	JCache implementation	Open source	ehcache.org
TestNG	TestNG	Java unit testing framework (JUnit-inspired)	Open source	testng.org/doc
The Grails Project	Grails	Groovy web framework (like Ruby on Rails)	Open source	grails.org
The Linux Foundation	Kubernetes	Container orchestration	Open source	kubernetes.io
The Netty Project	Netty	Event-driven, non-blocking JVM framework for protocol clients & servers	Open source	netty.io
Thinking Software	Race Catcher	Dynamic race detection	N/A	thinkingsoftware.com
ThoughtWorks	Go	Continuous delivery server	Open source	go.cd
Thymeleaf	Thymeleaf	Server-side Java web template engine	Open source	thymeleaf.org
Twilio	Twilio	Messaging APIs (text, voice, VoIP)	Available by request	twilio.com
Twitter	Finagle	RPC for high-concurrency JVM servers (Java & Scala APIs, uses Futures)	Open source	twitter.github.io/finagle
Twitter	Finatra	Scala HTTP services built on TwitterServer and Finagle	Open source	twitter.github.io/finatra
Vaadin	Vaadin	Server-side Java -> HTML5	Free tier available	vaadin.com
Vert.x	Vert.x	Event-driven, non-blocking JVM framework	Open source	vertx.io

COMPANY	PRODUCT	PRODUCT TYPE	VERTICAL	WEBSITE
vmlens	vmlens	Java race condition catcher	Open source version available	vmlens.com
Waratek	Waratek	Java security (runtime application self-protection)	Demo available by request	waratek.com
Wiremock	Wiremock	HTTP mocking	Open source	wiremock.org
WorldWide Conferencing	Lift	Scala web framework w/ORM, strong view isolation, emphasis on security	Open source	liftweb.net
WSO2	WSO2 Application Server	Web application server	Open source	wso2.com/products/application-server
WSO2	WSO2 Microservices Framework for Java	Microservices framework for Java	Open source	wso2.com/products/microservices-framework-for-java
XebiaLabs	XebiaLabs XL	Deployment automation & release management	30 days	xebialabs.com
Xstream	Xstream	XML serialization	Open source	x-stream.github.io
Yammer	Dropwizard	REST web services framework (opinionated, rapid spinup)	Open source	dropwizard.io/1.3.1/docs
YourKit	YourKit Java Profiler	Java CPU & memory profiler	Free developer version	yourkit.com
ZeroTurnaround	JRebel	Class hot-loading (in running JVM)	N/A	zeroturnaround.com/software/jrebel
ZeroTurnaround	XRebel	Java web app profiler	N/A	zeroturnaround.com
Zkoss	ZK Framework	Enterprise Java web framework	Open source	zkoss.org
Zoho	Site24x7	Website, server, APM	30 days	site24x7.com
ManageCat	ManageCat	Manage, monitor, & troubleshoot Apache Tomcat	Available by request	managecat.com
Yonita	Yonita	Automated detection of code defects	N/A	yonita.com

GLOSSARY

APPLICATION PROGRAM INTERFACE (API): A set of tools for determining how software components should act within an application.

CONCURRENCY: The ability to run several applications, or several parts of an application, at the same time.

CONVENIENCE FACTORY METHODS FOR COLLECTIONS: A new, more succinct and convenient, way to create immutable collections like List, Set, and Map in Java (added in Java 9).

COROUTINE: Coroutines simplify asynchronous programming by putting the complications into libraries. The logic of the program can be expressed sequentially in a coroutine, and the underlying library will figure out the asynchrony.

DESIGN PATTERN: A reusable, high-level solution to a common problem in an application or architecture.

DOMAIN-DRIVEN DESIGN (DDD): A software development practice in which an application's focus is on the domain or set of requirements or functionalities, and developers work with the business to ensure the application meets these requirements.

ECLIPSE OMR: Cross-platform components for building reliable, high-performance language runtimes.

ECLIPSE OPENJ9: A high-performance, scalable, Java virtual machine (JVM) implementation.

ELVIS OPERATOR: The Elvis operator `?:` is a binary operator in the Kotlin programming language that returns its first operands if that operand is null, and otherwise evaluates and returns its second operand.

ENTERPRISE ARCHITECTURE: The fundamental decisions about the way an enterprise application will be built that will be difficult to change afterward.

IMMUTABLE COLLECTIONS: Implementations of Collection or Map, which throw

Exceptions when any modifying method, e.g. `add()`, `put()`, `set()`, is called.

INITIALIZED VARIABLE: A variable that is assigned a value (usually whose type can be inferred) upon declaration.

INSTRUMENTATION: Hooks inside the JVM and/or your code to allow visibility into the inner workings.

JAKARTA EE: New project under Eclipse Foundation designed to replace Java EE; will continue to advance existing Java EE and open standard technologies that target server-side Java. Jakarta EE has a focus on cloud-native and is owned by the Eclipse Foundation.

JAVA DEVELOPMENT KIT (JDK): A free set of tools, including a compiler, provided by Oracle, the owners of Java.

JAVA ENTERPRISE EDITION (JAVA EE): A platform that provides an API for object-relational mapping, web services, and distributed architectures to develop and deploy Java apps.

JAVA VIRTUAL MACHINE (JVM): Abstracted software that allows a computer to run a Java program.

JAVA MANAGEMENT EXTENSIONS (JMX): Tools for monitoring and managing Java applications, networks, and devices.

JSHELL: A REPL introduced in Java 9 that lets developers experiment with code from the command line.

KOTLIN: A language that runs on the JVM, developed by JetBrains and provided under the Apache 2.0 License, offering both object-oriented and functional features.

LAMBDA EXPRESSIONS: An expression in Java 8 that allows base classes to be implemented without being named.

MEMORY LEAK: A resource leak that occurs when a computer program incorrectly manages memory allocations.

METRONOME GC: A soft, real-time garbage collector (GC) in Eclipse OpenJ9.

MICROPROFILE: A community-driven initiative that complements Java EE under the Eclipse Foundation; utilizes Java EE technologies and targets microservices applications on the cloud.

MICROSERVICES ARCHITECTURE: An architecture for an application that is built with several modular pieces that are deployed separately and communicate with each other, rather than deploying one single piece of software.

MODULE: A self-contained programming unit that exposes a specific set of APIs to the outside world and hides the rest. A module can also specify which other modules it requires and which other modules can use it.

OPENJ9 "ECO" MODE: Reduces CPU time when JVM is idle.

OPTIONAL: A Java SE container object that may or may not contain a not-null value. The class provides several methods that depend on the presence or absence of the contained value.

PRIVATE JRE: A Java runtime environment that ships with a Java application that contains only those libraries needed by an application. Private JREs are generally smaller than the standard JREs installed by the Java installer. Enclosing a private JRE with a Java application guarantees that the application can run, even if Java was not previously installed on the machine.

RESERVED TYPE NAME: A name of a type that has special meaning with a compiler and cannot be used as a user-defined class or interface name.

SCALA: An object-oriented programming language that runs on the JVM and is interoperable with Java and has many features of functional programming languages.

SERVICEABILITY: The term the JDK uses for instrumentation inside the JVM.

SPRING FRAMEWORK: An open-source collection of tools for building web applications in Java.

STATIC FIELD: A field that will be held in common by all instances of a class.

STREAM: A sequence of data that is read from a source and then written to a new destination.

TYPE INFERENCE: The ability to implicitly determine the type of an expression.

INTRODUCING THE

Open Source Zone

Start Contributing to OSS Communities and Discover Practical Use Cases for Open-Source Software

Whether you are transitioning from a closed to an open community or building an OSS project from the ground up, this Zone will help you solve real-world problems with open-source software.

Learn how to make your first OSS contribution, discover best practices for maintaining and securing your community, and explore the nitty-gritty licensing and legal aspects of OSS projects.

COMMITTERS & MAINTAINERS

COMMUNITY GUIDELINES

LICENSES & GOVERNANCE

TECHNICAL DEBT

Visit the Zone

BROUGHT TO YOU IN PARTNERSHIP WITH   FLEXEra