DZone

DZONE'S 2019 GUIDE TO

Java

NEW DEVELOPMENTS AND FEATURES

BROUGHT TO YOU IN PARTICIPATION WITH

WhiteSource

Dear Reader,

Despite reoccurring debate, Java is most certainly not dead. The JDK's newest release cycle and continued widespread use has proved that Java is still the number one choice for our main applications. And yet, the ecosystem is in the middle of major changes.

Last July, Oracle announced a new subscription-based pricing model that would require Java SE commercial users to acquire a special license in order to continue to receive support and updates. Oracle customers and personal users would continue to have access to the OpenJDK, and other JDKs based on it, but companies could no longer access open-source versions for business use.

Along with changes in pricing, the adjusted release cycle offers a new version of the JDK every six months. And, JDK 11, released in October 2018, is the most recent long-term support version, and best option for developers who want to upgrade from Java 8.

But, according to the 2018 JVM Ecosystem Survey, more than 88 percent of developers continue to use Java 8, despite newer versions. Many developers are hesitant about switching to Java 9 or higher due to fear of breaking existing applications. Additionally, developers are unable to find support in Java 9 or up for some of their tools and IDEs.

So, what does this mean for the future of the JDK?

Well, it means a lot of things. But most importantly, it means that Java will continue to exist across a wide variety of platforms and applications.

The existence of multiple Java versions and open-source movements have paved the way for new languages, tools, and platforms to emerge. Kotlin and Clojure, for example, are the second most popular JVM languages, after Java, and are being successfully implemented across a wide range of applications.

Based on the report, "Exactly 9 in 10 JVM users are using Java for their main applications. While many projects today are defined as multi-language or polyglot, the part on the JVM primarily runs Java."

In this guide, we focus on the latest tools and features in the JVM and how to best implement them in your existing Java projects. Open-source Java tools such as MicroProfile, which provides various microservices features to Java EE, and scaling apps in the cloud are some of the more exciting, and challenging, new ways to build upon your existing Java code.

Java is still free and continues to be the trusted language by developers all over the world — Java is not going anywhere. The release cycle will simply provide developers with faster updates and new features on a regular basis. And while change can be hard, it's an exciting time as we welcome the new era of Java.

Thanks for reading and we hope you enjoy!

WRITTEN BY LINDSAY SMITH
CONTENT COORDINATOR, DEVADA

Table of Contents

DZone is...

BUSINESS & PRODUCT
Matt Tormollen
CEO

Matt Schmidt
President

Terry Waters
Interim General Manager

Jesse Davis
EVP, Technology

Kellet Atkinson
Media Product Manager

SALES
Kendra Williams
Sr. Director of Media Sales

Chris Brumfield
Sales Manager

Jim Dyer
Sr. Account Executive

Tevano Green
Sr. Account Executive

Brett Sayre
Account Executive

Alex Crafts
Key Account Manager

Eniko Skintej
Key Account Manager

Craig London
Key Account Manager

Jordan Scales
Sales Development Rep.

MARKETING
Susan Wall
CMO

Aaron Tull
Dir. of Demand Gen.

Waynette Tubbs
Dir. of Marketing Comm.

Ashley Slate
Sr. Design Specialist

Colin Bish
Member Marketing Spec.

Suha Shim
Acquisition Marketing Mgr.

Cathy Traugot
Content Marketing Mgr.

PRODUCTION
Chris Smith
Director of Production

Billy Davis
Production Coordinator

Naomi Kromer
Sr. Campaign Specialist

Jason Budday
Campaign Specialist

Michaela Licari
Campaign Specialist

EDITORIAL
Matt Werner
Publications Coordinator

Sarah Davis
Publications Associate

Mike Gates
Content Team Lead

Kara Phelps
Content & Comm. Manager

Tom Smith
Research Analyst

Jordan Baker
Content Coordinator

Andre Lee-Moye
Content Coordinator

Lauren Ferrell
Content Coordinator

Lindsay Smith
Content Coordinator

Sarah Sinning
Staff Writer

Executive Summary

BY KARA PHELPS CONTENT & COMMUNITY MANAGER, DEVADA

Despite the rise of languages like Kotlin and Clojure in recent years, Java is still a powerhouse. It's one of the most popular programming languages in the world, with more than 9 million developers over the two decades and counting that it's been publicly implemented. This year, we asked 492 Java developers for their thoughts on the latest versions of the language, changing paradigms, the use of Java in their organizations, and much more.

JAVA 8 IS STILL THE FAVORITE

DATA

While developers may use several versions of Java depending on their organization or use case, 86 percent of survey respondents reported that they use Java 8 to build new applications. 24 percent use Java 11, 10 percent use Java 10, 9 percent use Java 9, 15 percent use Java 7, and 6 percent use Java 6 or below. Similarly, 83 percent of respondents said they use Java 8 in existing applications. Just 11 percent use Java 11, 5 percent use Java 10, and 7 percent use Java 9. 42 percent said they still use Java 7 in existing applications, and 25 percent said they use Java 6 or below.

IMPLICATIONS

Java 8 has now been out for five years, which has given enterprises some time to adopt it. Java 8 also included updates to improve application performance, as well as to make functional programming easier. Oracle released one last update to Java 8 in January 2019 and will support the release through December 2020. Despite these end-of-life preparations, though, developers still use Java 8 the most.

RECOMMENDATIONS

The wide use of Java 8 means that it's probably a good time to learn how to take advantage of the functional programming options that Java 8 provides, if you aren't already. 44 percent of survey respondents said they and their teams are "comfortable" mixing object-oriented and functional paradigms. 34 percent of respondents said they were "somewhat" to "very" uncomfortable, however.

OLD SCHOOL VS. NEW SCHOOL

DATA

61 percent of survey respondents said they have no opinion as to whether the "new style" Java 10 features make it more fun to program in Java. That's an increase from last year, when 49 percent said they had no opinion. This year, 32 percent of respondents replied "yes" to that question — decreasing from 44 percent who answered "yes" last year.

IMPLICATIONS

Interest in "new style" Java 10 features appears to be waning among survey respondents. Of course, a year has now passed since the release of Java 10, and we are now on to Java 12 (editor's note: Java 12 had not been released at the time the survey was taken), but 58 percent of survey respondents reported that they are not currently writing code in Java 10 or above. Improvements may get pushed out to developers more quickly with the new six-month release cadence, but we may be seeing the beginning of "release fatigue."

RECOMMENDATIONS

Old habits die hard (especially in the enterprise). It makes sense to continue using the Java features that keep your legacy applications running. 33 percent of survey respondents said they never mix "old style" and "new style" Java. However, it may be worth checking out newer features to solve stubborn issues. When asked where they mix "old" and "new," 33 percent of respondents said they do so with APIs for creating unmodifiable collections. 32 percent said they did so using the `optional.orElseThrow()` method, 30 percent said they remove deprecated methods, and 27 percent said they use the local variable type inference.

REACTING TO ORACLE'S JDK CHANGES

DATA

33 percent of survey respondents using Java 8 said they use Oracle's JDK and have no plans to switch due to the change in long-term support. Another 33 percent of Java 8 users said they are planning to move to a different distribution of the JDK, and 21 percent said they are using a different distribution already. Among Java 11 users, 4 percent said they use Oracle's JDK and don't plan to switch. 10 percent of these users plan to move to a different distribution, and 11 percent are already using one. (Java 12 had not been released at the time of the survey.)

IMPLICATIONS

Starting with the April 2019 update, Oracle JDK 8 will be restricted for commercial use, but there are plenty of alternative options. Many other providers are offering free OpenJDK builds, and starting with Java 9, Oracle has provided OpenJDK builds that are free for commercial use (although they're only updated for six months). A higher percentage of Java 8 users seem to know how they'll be managing the transition.

RECOMMENDATIONS

"Java is still free," according to a prominently shared Google doc by the Java Champions describing Oracle's changes in long-term support for its JDK. They're right — the open-source options for Java 8 and 11 have full capabilities for commercial use, and personal use will continue to be free as well.

Key Research Findings

BY JORDAN BAKER
CONTENT COORDINATOR, DEVADA

Demographics

For this year's DZone Guide to Java survey, we received 804 total responses, with 492 complete responses. Below is the basic demographic breakdown of the respondents.

- Respondents have an average of 16 years of experience in IT.
- Respondents live in four main geographical areas:
  - 36% in Europe
  - 18% in South Central Asia
  - 15% in the USA
  - 12% in South America
- Survey-takers reported working in three main industry verticals:
  - 22% work for software vendors
  - 15% work in finance/banking
  - 12% work in consulting
- A majority of respondents work for enterprise-level organizations.
  - 27% for organizations sized 100-999
  - 21% for organizations sized 10,000+
  - 20% for organizations sized 1,000-9,999
- Three main roles were reported:
  - 42% work as developers/engineers
  - 24% work as developer team leads
  - 19% work as architects
- Respondents reported working on four main types of software projects:
  - 83% develop web applications
  - 49% work on enterprise business apps
  - 24% develop native mobile applications
  - 15% work on high-risk software

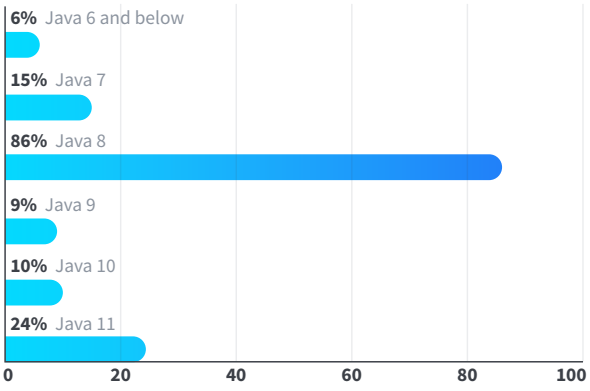
Java 8 Still the Favorite

Despite the fact that it's eight years old, Java 8 remains the most used and preferred version of the Java language. When we asked respondents what versions of Java are being used by their organization to work on existing applications, 83% reported using Java 8, while Java 7 came in a distant second with a 42% adoption rate. When we compare these statistics to our data on the two types of software most respondents develop (web applications and enterprise business applications) as delineated in the Demographics section, we find that Java 8, though dominant among both groups, proves slightly more popular among developers/organizations working on enterprise business apps. Among respondents who reported developing web apps, 82% said they use Java 8 on existing applications, while 87% of those who reported developing enterprise business apps told us they use Java 8 for existing applications.

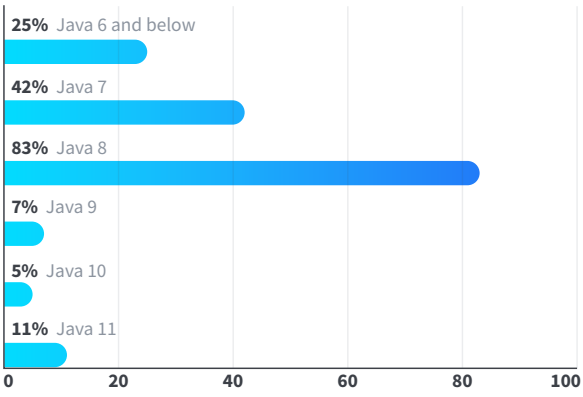
Given Java 8's age, its dominant position in existing applications is less surprising than its widespread use in building new applications. 86% of survey-takers reported using Java 8 to create new applications, while only 24% said they use Java 11, 10% use Java 10, and a mere 9% use Java 9. If we compare our data on Java 8 usage rates against our data on respondents developing

SURVEY RESPONSES

What versions of Java are being used at your organization when building new apps?



What versions of Java are being used in existing apps at your organization?



web and enterprise business applications, we see a similar trend to that reported above. Among respondents working to develop web applications, 87% use Java 8 to create their new apps, with 24% use Java 11, 10% use Java 10, and 10% use Java 9. For those respondents working to develop enterprise business apps, we found that 90% use Java 8 in the building of new applications, while, again, 24% use Java 11, 10% use Java 10, and only 7% use Java 9.

Even with the extreme popularity of Java 8, one would expect the newer versions of the language (i.e. Java 10 and 11 as well as the soon-to-be-released Java 12) to garner some excitement from a developer community that's so interested in Java. Yet, a majority of respondents seemed to be lukewarm in their opinions of the newer versions of Java and the features these versions brought with them. When we asked respondents what they thought constituted the most important new feature introduced in Java 10, well over half (70%) reported that they had no opinion on the matter. We also asked survey-takers which features of Java 11 they were using, and five features of Java 11 (Epislon, ZGC, Flight Recorder, `readString()` and `writeString()`, and the new HTTP client) all had non-adoption rates (i.e. are not being used) of 80% or higher. And finally, when asked about the new features in Java 12 for which they were excited, 53% of respondents reported "none of the above."

Despite this general lack of enthusiasm around the newer versions of Java, over 60% of survey-takers seem excited for the future of the language. 43% told us they feel fairly optimistic about Java's future, and another 29% feel very optimistic.

Development Practices and Coding Paradigms

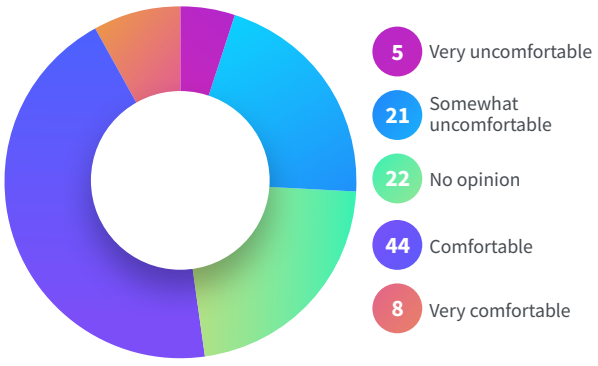
Object-oriented and functional programming represent two of the more popular coding paradigms in use today. This is reflected in that fact that 44% of respondents reported feeling "comfortable" using and mixing these two paradigms and another 8% asserted feeling "very comfortable" with this practice. Interestingly, it appears that respondents' comfort level in mixing object-oriented and functional programming varied by the types of applications they develop. Among those respondents working on web applications, 45% reported feeling comfortable mixing the object-oriented and functional paradigms, while 9% said they were very comfortable. Additionally, 21% of respondents working as web app developers told us there were "somewhat uncomfortable" mixing these two paradigms, while 4% reported being "very uncomfortable." But among those respondents developing enterprise business applications, 35% reported feeling comfortable mixing object-oriented and functional coding paradigms, while 5% reported feeling very comfortable. On the other side of the coin, 28% of these respondents said they feel somewhat uncomfortable in this practice, while 10% feel very uncomfortable. One reason for enterprise business app devs' reluctance to mix coding paradigms could come from the fact they work with massive code bases filled with legacy code that takes advantage of older coding paradigms. Thus, introducing object-oriented or functional paradigms (or a mix of the two) could result in a need for large refactoring projects.

When it comes to development practices specific to the Java language, however, a greater level of parity appears among the responses. When asked how they mix "old" and "new" style Java code, 33% reported using APIs for creating unmodifiable collections, another 33% said they never mix "old" and "new" Java, and 32% said they continue to use the `optional.orElseThrow()` method in their code. Additionally, 30% of respondents told us they mix "old" and "new" style Java while working on the removal of deprecated methods. This widespread mixing of "old" Java standards makes sense given the continued dominance of Java 8.

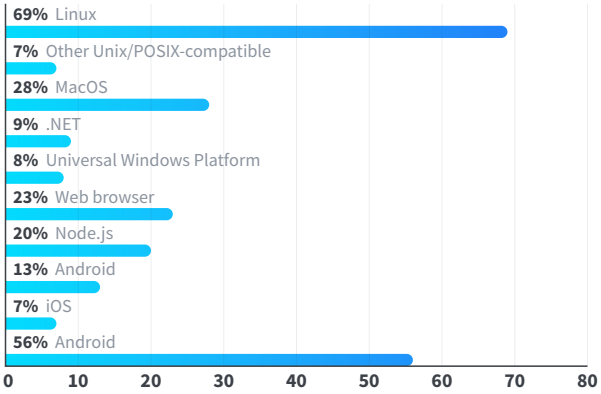
The JDK, no matter the provider, has become an integral part to any Java developer's work. It's interesting to see that the JDK developers use seems to be dependent on the version of Java they're currently working with. We asked respondents if they were

SURVEY RESPONSES

How comfortable are you and your team at mixing object-oriented and functional paradigms?



Which of the following platforms/runtimes/operating systems do you primarily develop on?



planning to stop using Oracle's JDK due to the change in Oracle's JDK long-term support and saw an interesting variance in answers. Among users of Java 10 and 11, the most popular response was, "I am using a different distribution of the JDK." Among respondents using all other versions of Java (i.e. Java 6-9), the most popular responses were, "I am planning to move to a different distribution of the JDK" and "I use Oracle's JDK and have no plans to switch." Thus, while users of every iteration of Java have some interest in exploring non-Oracle SDKs, it seems that those developers using the newer versions of the language have already jumped the preverbal Oracle SDK ship.

Java Adjacent

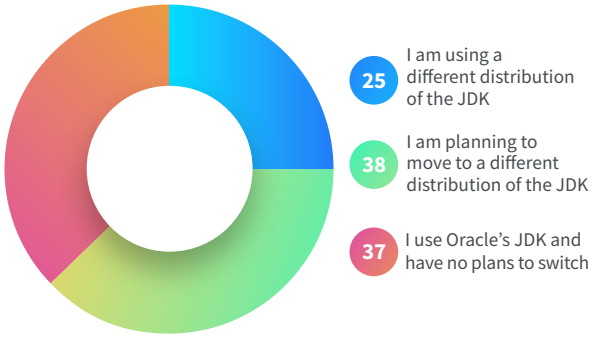
Despite all the features we've discussed so far, Java and the applications built upon it cannot operate in a vacuum. Thus, in this section, we'll focus on how developers use Java alongside other important technologies. To start off, we found that respondents had two main platforms/runtimes/operating systems on which they primarily develop. 69% of respondents told us they typically develop on Linux-based operating systems, while 56% reported using Win32/64. When we compare this data to our data on web application and enterprise business app developers, we find that these two systems remained, far and away, the most popular platforms/runtimes/operating systems to develop on. 72% of web application developers reported using Linux and 54% use Win32/64, while 69% of enterprise business app developers develop on Linux and 63% use Win32/64. Interestingly, when we asked respondents which of these platforms/runtimes/operating systems they target, we saw a bigger discrepancy between the answers of web app and enterprise business app developers. Among the general survey population, 84% of respondents target Linux and 42% target Win32/64. When we narrow this down to respondents working on web applications, we find that 86% target Linux and 38% target Win32/64; and among respondents working on enterprise business applications, 82% target Linux and 49% target Win32/64.

Not all technologies in the survey witnessed such variance, however. Indeed, when we asked which API quality attributes are most important, we saw almost mirrored results across three respondent groups. Among the general survey population, 74% told us they care about simplicity, 60% said consistency, 60% reported performance/robustness, 58% feel productivity is important, and 54% care about learnability. When we again pare the data down to reflect those respondents working on web applications and enterprise business applications, we see almost the exact same results. Among web developers, the most important API quality attributes were simplicity (74%), consistency (64%), performance/robustness (62%), productivity (59%), and learnability (54%). Among enterprise business developers, the most important API qualities reported were simplicity (75%), performance/robustness (66%), productivity (61%), consistency (59%), and learnability (53%). The only notable variance among all these factors is how performance/robustness appears more important for respondents working on enterprise-level applications.

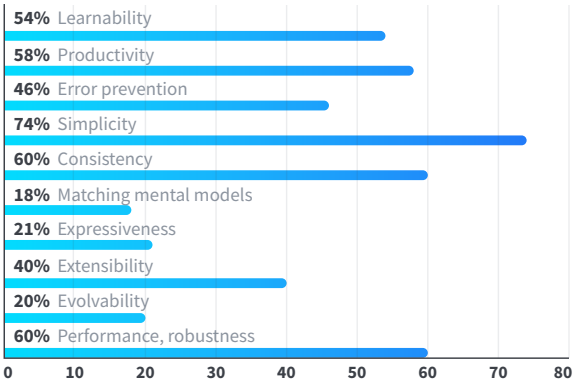
To round out this report, let's examine the use of microservices among Java developers. Of those surveyed, 58% reported using microservices in some capacity during development, while another 18% said they do not, but are planning on adopting microservices. Of note, microservices proved more popular among those respondents working on web applications than those working on enterprise business apps. Of those respondents currently developing web apps, 63% told us they are currently using microservices in some manner, while 18% said they are not using microservices but are planning to in the future. Among respondents currently working on enterprise business applications, we found that 54% use microservices in their development efforts, and 22% are planning on adopting microservices.

SURVEY RESPONSES

Are you planning to stop using Oracle's JDK due to the change in Oracle's JDK long-term support?



Which of the following API quality attributes are most important for you (for any API, not just web APIs)?



Diving Deeper Into Java

Zones

Java dzone.com/java

The largest, most active Java developer community on the web. The Java Zone provides 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.

Twitter

 @trisha_gee

 @rafabene

 @brunoborges

 @dianecraigdavis

 @jessitron

 @sugrue

 @danielbryantuk

 @mon_beck

 @lunivore

 @s1m0nw1

Refcardz

Java Application Vulnerabilities

Java applications, like any other, are susceptible to gaps in security. This Refcard focuses on the top vulnerabilities that can affect Java applications and how to combat them.

Java Containerization

This Refcard focuses on the design, deployment, service discovery, and management of Java applications on the open-source project called Docker so that you can get your Java application up and running inside a Docker-deployed Linux container.

Java API Best Practices

In this Refcard, you'll learn how to make well-documented, consistent, evolvable APIs to help your users make the most of your Java applications.

Books

Effective Java

Learn about the latest language and library features that Java has to offer, get insight into Java subtleties, and see how modern Java supports multiple paradigms.

Java Concurrency in Practice

Dive into basic concepts of concurrency and thread safety, techniques for building and composing thread-safe classes, using concurrency building blocks, and more.

Modern Java in Action

Explore the new features of Java 9 like support for reactive programming and learn how to build on your existing Java skills with the newest techniques.

Podcasts

Java Pub House

Learn about real issues that developers face when programming Java, like O/R setups, threading, troubleshooting, and more.

The Changelog

Through conversations with hackers, leaders, and innovators, learn about OSS code, service meshes, cloud-enabled apps, and more.

Software Defined Talk

Dive into various topics of interest to Java developers, like Kubernetes, serverless, cloud, DevOps, and coding.

Java Garbage Collection

QUICK VIEW

01. Java 11 and 12 added new garbage collectors that can help you test and run your code

02. The Epsilon GC lets you find out what happens if you run your Java code with no garbage collector.

03. Shenandoah and ZGC are two new garbage collectors for Java apps that need lots of memory.

BY ERIC GOEBELBECKER
SERVER-SIDE SOFTWARE ENGINEER

Java manages heap memory for you. You run your application inside a virtual machine (JVM). The JVM does the work of allocating space when necessary and freeing it when it is no longer needed. The garbage collector (GC) is what does this work for you.

Recycling memory involves garbage collection "cycles" that have an impact on performance. How much impact depends on the nature of your application and the GC you choose. The good news is that there's more than one garbage collector, and each recent release of Java has offered more choices. You can pick the GC that best suits your application's needs.

New Garbage Collectors

The last two releases of Java introduced three new GCs. We'll take a look at each of them and what they add to the ecosystem.

Java 11 introduced Epsilon and the Z garbage collector (ZGC). Epsilon is the "no-op" garbage collector. It allocates new memory but never recycles it. ZGC promises to manage vast amounts of memory with high throughput and short pause times.

Java 12 adds the Shenandoah Garbage Collector. Like ZGC, it manages large heaps with short pause times but uses a very different approach.

Epsilon Garbage Collector

WHAT IS EPSILON?

Epsilon is a passive or "no-op" GC. It handles memory allocation but doesn't recycle it when objects are no longer used. When your application exhausts the Java heap, the JVM shuts down. In other words, Epsilon will allow your application to run out of memory and crash.

HOW TO USE EPSILON

COMMAND LINE OPTIONS	NOTES
`-XX:+UnlockExperimentalVMOptions`	Unlock Java experimental options.
`-XX:+UseEpsilonGC`	Use Epsilon GC.
`-XmxXg`	Set heap size. JVM will exit with an `OutOfMemoryError` if this amount is exceeded.
`-XX:HeapDumpOnOutOfMemoryError`	Generate a heap dump if JVM runs out of memory.
`-XX:OnOutOfMemoryError=`	Run specified command when an out-of-memory error occurs.

WHY USE A NO-OP GC?

A no-op garbage collector is useful for measuring and managing application performance. Active garbage collectors are complex programs that run inside the JVM alongside your application. They incur overhead that can cause latency and reduce throughput.

Epsilon removes the impact GC has on performance. There are no GC cycles or read or write barriers. When using the Epsilon GC, your code runs in isolation. You can use it to see how garbage collection affects your app's performance and what your memory threshold is since it'll tell you when it runs out. If you think you only need four gigabytes of memory, run it with `-Xmx4g` and see if you're right. If you're wrong, rerun it with `XX:HeapDumpOnOutOfMemoryError` enabled and take a look at the heap dump to see where you're wrong.

If you need to squeeze every bit of performance out of your application, Epsilon might be your best option for a GC. But you need to have a complete understanding of how your code uses memory. If it creates almost no garbage or you know exactly how much memory it uses for the period it runs in, Epsilon is a viable option.

The Z Garbage Collector

WHAT IS THE Z GARBAGE COLLECTOR?

ZGC is a low-latency GC designed to work well with huge amounts of memory. The Oracle documentation refers to multi-terabyte heaps in its description of Z. Oracle introduced ZGC in Java 11. In Java 12, Oracle added performance fixes and class unloading even though Z is still in experimental status. It's only available on 64-bit Linux.

HOW DOES ZGC WORK?

ZGC works concurrently with your application, performing all its work in its threads. It uses load barriers for heap references. Load barriers cause fewer delays than those imposed by the G1 collector's pre- and post-write barriers.

ZGC takes advantage of 64-bit pointers with a technique called pointer coloring. Colored pointers store extra information about objects on the heap. (This is one of the reasons it's limited to the 64-bit JVM.) By limiting the GC to 4TB heaps, the developers have 22 extra bits in each pointer to encode additional information. Z uses four extra bits at the moment. Each pointer has a bit for `finalizable`, `remapped`, `mark0`, or `mark1`.

The Z garbage collector remaps objects when memory becomes fragmented. The mapping avoids the performance hit incurred when the GC needs to find space for a new allocation. Pointer coloring helps with remapping since a remapped reference discovers the new location at the next access.

When your application loads a reference from the heap, ZGC checks the extra bits. If it needs to do any extra work (e.g. getting a remapped instance), it handles it in the load barrier. It only has to do this once, when it loads the reference. This sets it apart from the write barriers used by mainline garbage collectors like G1.

The Z garbage collector performs its cycles in its threads. It pauses the application for an average of 1 ms. The G1 and Parallel collectors average roughly 200 ms.

HOW TO USE ZGC

COMMAND LINE OPTIONS	NOTES
`-XX:+UnlockExperimentalVMOptions`	Unlock Java experimental options.
`-XX:+UseZGC`	Use ZGC.
`-XmxXg`	Set heap size.
`-XX:ConcGCThreads=X`	Set number of GC threads.

ZGC is a concurrent garbage collector, so setting the right heap size is very important. The heap must be large enough to accommodate your application but also needs extra headroom so Z can meet new requests while relocating active objects. The amount of headroom you need depends on how quickly your application requests new memory.

ZGC will try to set the number of threads itself, and it's usually right. But if ZGC has too many threads, it will starve your application. If it doesn't have enough, you'll create garbage faster than the GC can collect it.

WHY USE ZGC?

ZGC's design works well with applications large heap sizes. It manages these heaps with pause times under 10ms and little impact on throughput. These times are better than G1's.

ZGC does its marking in three phases. The first is a short stop-the-world phase. It examines the GC roots, local variables that point to the rest of the heap. The total number of these roots is usually minimal and doesn't scale with the size of the load, so ZGC's pauses are very short and don't increase as your heap grows.

Once the initial phase completes, ZGC continues with a concurrent phase. It walks the object graph and examines the colored pointers, marking accessible objects. The load barrier prevents contention between the GC phase and any application's activity.

After ZGC has completed marking, it moves live objects to free up large sections of the heap to make allocations faster. When the relocation phase begins, ZGC divides the heap into pages and works on one page at a time. Once ZGC finishes moving any roots, the rest of the relocation happens in a concurrent phase.

ZGC's phases illustrate how it manages large heaps without impacting performance as application memory grows.

Shenandoah

WHAT IS SHENANDOAH?

Shenandoah is another garbage collector with low pause times. These times are short and predictable, regardless of the size of the heap. Shenandoah was developed at Red Hat and has been around for several years. It's now part of the Java 12 release.

Like ZGC, Shenandoah does most of its work in parallel with the running application. But its approach to garbage collection is different. Shenandoah uses memory regions to manage which objects are no longer in use and which are live and ready for compression. Shenandoah also adds a forwarding pointer to every heap object and uses it to control access to the object.

HOW DOES SHENANDOAH WORK?

Shenandoah's design trades concurrent CPU cycles and space for pause time improvements. The forwarding pointer makes it easy to move objects, but the aggressive moves mean Shenandoah uses more memory and requires more parallel work than other GCs. But it does the extra work with very brief stop-the-world pauses.

SHENANDOAH PHASES

Shenandoah processes the heap in many small phases, most of which are concurrent with the application. This design makes it possible for the GC to manage a large heap efficiently.

The first phase contains the first stop-the-world pause in the cycle. It prepares the heap for concurrent marking and scans the root set. Like ZGC, the length of this pause corresponds to the size of the root set, not the heap. Next, a concurrent phase walks the heap and identifies reachable and unreachable objects.

The third finishes the process of marking by draining pending heap updates and re-scanning the root set. This phase triggers the second stop-the-world pause in the cycle. The number of pending updates and size of the root set determine how long the pause is.

Then, another concurrent phase copies the objects out of the regions identified in the final mark phase. This process sets Shenandoah apart from other GCs since it aggressively compacts the heap in parallel with application threads.

The next phase triggers the third (and shortest) pause in the cycle. It ensures that all GC threads have finished evacuation. When it finishes, a concurrent phase walks the heap and updates references to objects moved earlier in the cycle.

The last stop-the-world pause in the cycle finishes updating the references by updating the root set. At the same time, it recycles the evacuated regions. Finally, the last phase reclaims the evacuated regions, which now have no references in them.

HOW TO USE SHENANDOAH

COMMAND LINE OPTIONS	NOTES
`-XX:+UnlockExperimentalVMOptions`	Unlock Java experimental options.
`-XX:+UseShenanodoahC`	Use Shenandoah GC.
`-XmxXg`	Set heap size.
`-XX:ShenandoahGCHeuristics=`	Select heuristics.

SHENANDOAH HEURISTICS

You can configure Shenandoah with one of three heuristics. They govern when the GC starts its cycles and how it selects regions for evacuation.

1. **Adaptive**: Observes GC cycles and starts the next cycle so it completes before the application exhausts the heap. This heuristic is the default mode.
2. **Static**: Starts a GC cycle based on heap occupancy and allocation pressure.
3. **Compact**: Runs GC cycles continuously. Shenandoah starts a new cycle as soon as the previous finishes or based on the amount of heap allocated since the last cycle. This heuristic incurs throughput overhead but provides the best space reclamation.

FAILURE MODES

Shenandoah needs to collect heap faster than the application it's serving allocates it. If the allocation pressure is too high and there's not enough space for new allocations, there will be a failure. Shenandoah has configurable mechanisms for this situation.

- **Pacing**: If Shenandoah starts to fall behind the rate of allocation, it will stall allocation threads to catch up. The stalls are usually enough for mild allocation spikes. Shenandoah introduces delays of 10ms or less. If pacing fails, Shenandoah will move to the next step: degenerated GC.
- **Degenerated GC**: If an allocation failure occurs, Shenandoah starts a stop-the-world phase. It uses the phase to complete the current GC cycle. Since a stop-the-world doesn't contend with the application for resources, the cycle should finish quickly and clear the allocation shortfall. Often, a degenerated cycle happens after most of the cycle's work is already completed, so the stop-the-world is brief. The GC log will report it as a full pause, though.
- **Full GC**: If both pacing and a degenerated GC fail, Shenandoah falls back to a full GC cycle. This final GC guarantees the application won't fail with an out-of-memory error unless there's no heap left.

WHY USE SHENANDOAH?

Shenandoah offers the same advantages as ZGC with large heaps but more tuning options. Depending on the nature of your application, the different heuristics may be a good fit. Its pause times might not be as brief as ZGC's, but they're more predictable.

While Shenandoah was not made available as part of Java until version 12, it's been around longer than ZGC. It's seen more testing and is even available as a backport for both Java 8 and 10.

Conclusion

Java GCs have come a long way in a few short years. The default G1 collector has made significant improvements since Java 7, and three new ones have been added to the platform.

Depending on your needs, one of these new GCs may be what you need to take your Java application performance to the next level.

ERIC GOEBELBECKER is an experienced software engineer with more than 25 years of experience working with market data, financial transactions, and news. Outside of work, he enjoys training dogs and practicing martial arts. He enjoys writing for technical blogs as an author for Hit Subscribe, an online content agency. LinkedIn Twitter

How to Use Recent MicroProfile and JDK Features to Scale Your Apps in the Cloud

QUICK VIEW

01. Learn how to launch single-file source code programs and a standard HTTP to client help you build Java into your cloud applications.

02. See how to work around the removal of XML APIs when working with MicroProfile and Spring apps.

03. Learn about building fault tolerance and health checks into your microservices without code.

BY BRIAN BENZ
SENIOR CLOUD DEVELOPER ADVOCATE AT MICROSOFT

Managing a potentially unlimited number of autonomous, loosely coupled, and frequently updated microservices on top of a flexible cloud infrastructure creates new challenges for developers. This article outlines features of recent JDK and MicroProfile updates for reliably deploying and scaling cloud applications while continuously integrating network and service updates. We'll also cover tips for enabling documentation, authentication, and managing decoupled application dependencies, resources, and configuration.

Adding Notable JDK Updates to Your Microservices

The latest JDK release has an interesting collection of new features that can help with building microservice-based applications. JDK 11 has been generally available since September 2018 and is the first long-term support version after Java 8. As always, features and scheduled releases of the latest JDKs are managed via the JDK Enhancement Proposal (JEP) Process. I've included the JEP release numbers and links as a handy reference to the full details of each new feature.

Launching Single-File Source Code Programs and a Standard HTTP Client

The ability to launch single-file source code programs (JEP 330), and the new standard HTTP Client (JEP 321), goes hand-in-hand with microservices, and makes things simple and easy to follow.

The ability to launch single-file programs enables quick, local testing of a developer's code and can also greatly simplify the instructions needed to run the code in a container-based environment. Note this sentence in the motivation section of JEP 330:

"Single-file programs — where the whole program fits in a single source file — are common in the early stages of learning Java and when writing small utility programs."

Of course, microservices could be described as small utility programs. And you can create a container to load a JDK image in this context. Then, you just add a `COPY` to load the file into the container and a CMD to get it started in your Dockerfile:

```
FROM adoptopenjdk/openjdk11:latest
COPY mymicroservice.java / mymicroservice.java
CMD ["java", "mymicroservice.java"]
```

Now, imagine how this simplifies testing. Instead of building/compiling and running your code in a container, you can run it in different JVM and OS versions by simply changing the `FROM` line in your Dockerfile!

Next up in your single-file microservice, you'll likely need to speak to other microservices, with at least a call and response, probably

using HTTP. To accomplish this in a very simple and readable way, implement the new standard HTTP Client (JEP 321) as part of your microservice. An `HttpClient` object sends requests via asynchronous requests, HTTP/2, and web sockets. An `HttpRequest` object contains a request method and headers, and an `HttpResponse` object returns information from the target. Here's a great example of a simple HTTP client implementation from Anthony Bruno with a comparison of the `URLConection` object it replaces.

Working Around the Removal of XML APIs When Working With MicroProfile and Spring Apps

Unfortunately, JEP 320 removes some handy XML processing classes as part of the removal of older Java EE and CORBA Modules, specifically JAX-WS (Java API for XML-Based Web Services) and JAXB (Java Architecture for XML Binding), which are common for reading and managing XML configuration files, among other functions.

Of course, you can build and/or refactor your apps to satisfy the Java world's current YAML fetish if that's your thing, but if you want to keep your applications as-is and use the latest JDK, the reference implementations of JAX-WS and JAXB are available as Maven artifacts:

- com.sun.xml.ws: jaxws-ri (JAX-WS, plus SAAJ and Web Services Metadata)
- com.sun.xml.bind : jaxb-ri (JAXB)

There are also Maven artifacts for tools and APIs; check the JEP 320 doc for full details and links.

Use Flight Recorder and Distributed Tracing to Collect and Analyze Telemetry in Your Microservices

Two recent developments can greatly help you plan and manage deployments in the cloud.

Distributed tracing can help manage and track microservice performance as well as the interactions between microservices. Distributed traces help manage and identify bottlenecks in your cloud applications, even if they're not all running on the same cloud, with minimal overhead and open-source visualization offerings.

From OpenTracing.io:

OpenTracing is comprised of an API specification, frameworks, and libraries that have implemented the specification and documentation for the project. OpenTracing allows developers to add instrumentation to their application code using APIs that do not lock them into any one particular product or vendor.

Flight Recorder (JEP 328) takes the analysis one step further and dials into the JVM to track performance bottlenecks and errors in the running code.

From the JEP 328 document:

"Flight Recorder records events originating from applications, the JVM, and the OS. Events are stored in a single file that can be attached to bug reports and examined by support engineers, allowing after-the-fact analysis of issues in the period leading up to a problem. Tools can use an API to extract information from recording files."

Flight Recorder is great in dev and test to manage the performance and reliability of the code. Distributed tracing is great to keep an eye on deployed microservices to see where bottlenecks are occurring across distributed microservices.

It's truly amazing to see all of the community effort that has gone into the JEP process and the Eclipse MicroProfile project.

Using MicroProfile to Manage Your Microservices

MicroProfile has been rapidly evolving as a great way to manage microservices with a few basic components that most microservices need. It's an Eclipse project based on Java EE 8 and has participation from major cloud vendors, IT organizations, and individuals. You can get more information on the MicroProfile project, components, and participants at microprofile.io. Also, check out the new MicroProfile starter page here.

Manage MicroProfile Configurations at Build and Runtime

One of the great features of MicroProfile is the configuration options. MicroProfile Config enables configurations externally and at runtime, meaning you can manage and even change some behaviors of an application based on the application environment without any changes to the application code.

You can use a pom.xml file to specify and transport configurations,

or it can be specified or overridden at runtime. System properties are evaluated first, then environment variables, then the pom.xml or project-defaults.yml files on the classpath. This enables not only flexible runtime environments but also easy testing of different configuration options via ENV vars and system properties.

For example, distributed tracing is one of the key components built into MicroProfile applications. To include Jaeger, an OpenTracing implementation and visualization tool in your applications, you just add this dependency to your pom.xml:

```
<dependency>
     <groupId>io.thorntail</groupId>
     <artifactId>jaeger</artifactId>
</dependency>
```

If you want to create a custom name for the service, you can add it in the project-defaults.yml files on the classpath like this:

```
swarm.jaeger.-jaeger-example
```

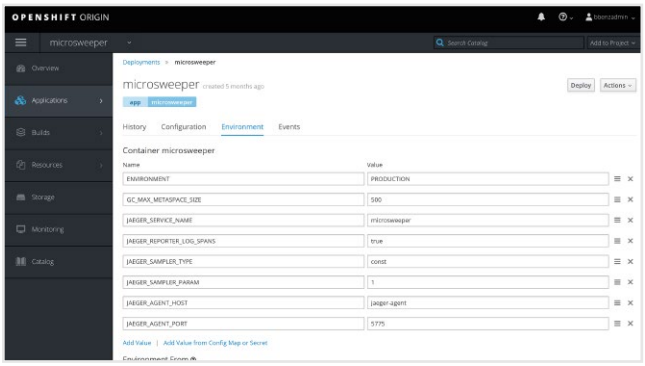
You can also set the name as an environment variable:

```
export -jaeger-example
```

If you're running a MicroProfile application with `Java -jar`, you can also specify settings via parameters on the command line:

```
Java -jar -jaeger-example
```

Some application platforms support environment variables for each deployment. Here's an example of Red Hat OpenShift Origin with Jaeger environment variables set for a deployed container in a production environment:



This enables major flexibility when building, testing, and deploying your apps in multiple cloud environments with loosely coupled services.

Building Fault Tolerance and Health Checks Into Your Microservice Without Code

Even with all the facilities that I've shared so far for managing code and testing, things can still go wrong in production. For those rare cases, you want to have easy ways to shut down and redeploy apps in a container environment. You also want to use similar features to help with scale by monitoring application load and adjusting the number of containers available based on user needs.

Health checks are used to check if a specific microservice is healthy before engaging it via another microservice or process. Fault tolerance is a way to manage what happens if a microservice is failing inside a larger infrastructure of multiple processes and microservices, and what automated action to take based on predetermined rules.

The good news is that, with MicroProfile, fault tolerance and health checks can be built into your application with no changes to the code whatsoever.

For example, to enable health checks in a MicroProfile app, just add this segment to the configuration section of the application's pom.xml file:

```
<config>
     <thorntail-v2-health-check>
     <path>/health</path>
     </thorntail-v2-health-check>
</config>
```

Once fault tolerance and health checks are enabled, they can be monitored and automated responses to predetermined events can be triggered with separate code. That code uses CDI to apply RetryPolicy, Fallback, BulkHead, and CircuitBreaker using annotations.

For more details check out the Health Check project and the Fault Tolerance project.

Conclusion

We live in interesting times! It's truly amazing to see all of the community effort that has gone into the JEP process and the Eclipse MicroProfile project. Most of the work represents amazing innovations in managing cloud-based microservices. If I've inspired you or you have any questions, please let me know — I love hearing from readers!

BRIAN BENZ is a Senior Cloud Developer Advocate at Microsoft, helping developers get the most out of Azure. Before Joining Microsoft, he was a solution architect, consultant, developer, author, and presenter at IBM, Deloitte, and other companies. LinkedIn Twitter

Find and Fix Java Open Source Vulnerabilities

Java is consistently one of the most popular languages in use today, providing us with beloved projects including Android SDK, the Spring Framework, and more.

However, despite its continued popularity, Java is witnessing a spike in the number of published vulnerabilities in open source components. WhiteSource's database shows that reported vulnerabilities for open source Java components more than doubled in 2018, rising from 187 in 2017 to 446 in 2018, and posing a higher risk to using them in our products.

The good news here is that the jump in the number of known vulnerabilities is due to the open source community doing a stellar job of finding and reporting these vulnerabilities, sharing their knowledge to help keep us safe.

On the flip side, we know that hackers love known vulnerabilities, scouring the NVD, security advisories, and other publicly available resources for intel on how to exploit these highly reusable open-source software components.

Your time is too valuable to be spent manually finding and fixing vulnerabilities, so why let vulnerability management get in the way? To solve this, there are developer-friendly tools that can integrate into your native coding environment and harness the widest range of security resources in order to keep you up to date on all vulnerabilities. This allows you to speed up remediations with suggested fixes, dramatically reducing the amount of work needed to stay secure, and — most importantly — allowing you to focus on coding.

WRITTEN BY RAMI SASS
CEO & CO-FOUNDER, WHITESOURCE

PARTNER SPOTLIGHT

WhiteSource

Helps you find and fix open source vulnerabilities in your code by fully integrating into your software development lifecycle.

Category Application Security | Software Composition Analysis | DevOps | Secure Coding

New Release Continuous

Open Source? Yes

Case Study Due to the sensitive regulatory environment Siemens Healthineers work in, Siemens H. needed assurance that they are remaining compliant and secure in their use of open source components.

"With open source software, usually the source code is available for all to see, including hackers," explained Code Clinic Lead, Neil Langmead. In order to avoid costly mistakes that can result from vulnerable or risky open source components being added to their products, Siemens Healthineers turned to WhiteSource.

"We chose WhiteSource because of its ease of use, its excellent data, and for the in-depth security vulnerability information that comes with the reporting engine." With WhiteSource, Siemens was offered the widest coverage of plugins and languages that they needed, as well as the continuous monitoring and policy enforcement safeguards they required in order to allow their team to code with confidence. Read more here

Strengths

- **Largest Vulnerabilities Database:** Continuously aggregates information from multiple sources
- **Comprehensive Coverage:** Supports 200+ programming languages and 20+ environments (incl. containers)
- **Pinpoint Accuracy:** Proprietary algorithms guarantee no false positives
- **Automated Workflow:** Enforce policies automatically at all stages of the SDLC
- **Easy Remediation:** Pinpoints vulnerable methods affecting your products

Notable Customers

- Microsoft
- Comcast
- EllieMae
- IGT
- Spotify

Website
whitesourcesoftware.com

Twitter
@WhiteSourceSoft

Blog
resources.whitesourcesoftware.com

MicroProfile: What You Need to Know

QUICK VIEW

01. MicroProfile was born out of a need to bring microservices-centric features to Java EE in order to accelerate the growth of the ecosystem.

02. Technologies based on MicroProfile include Open Tracing, JAX-RS, Fault Tolerance, and more.

03. This article will review the history of MicroProfile, its road map, the latest advancements, and a review of several MicroProfile features.

BY REZA RAHMAN
PRINCIPAL PROGRAM MANAGER FOR JAVA ON AZURE, MICROSOFT

MicroProfile is an open-source specification that brings a number of microservices-centric features to the Java EE ecosystem. It was born out of the need to accelerate Java EE innovation at a time when Oracle appeared to stop all Java EE 8 development. MicroProfile is already part of the Eclipse Foundation and very likely serves as an incubator to eventual standardization via Jakarta EE proper (Oracle donated Java EE to the Eclipse Foundation as Jakarta EE shortly after Java EE 8 was released).

The graphic below shows current MicroProfile technologies. MicroProfile is supported by major industry players like IBM, Red Hat, Oracle, Microsoft, Payara, and Tomitribe. Some notable MicroProfile API implementations include Payara, WebSphere Liberty, Quarkus, Helidon, KumuluzEE, and TomEE.

This article is a brief look at the current state of MicroProfile. This includes history, latest changes, road map, and a code example-driven look at features.

History

One of the key tenets of MicroProfile has been rapid release cycles, especially as compared with Java EE in recent years. The following chart shows the MicroProfile release history:

VERSION	DATE	CHANGES
1.0	September, 2016	Initial release with JAX-RS 2.0, JSON-P 1.0, and CDI 1.2 (parts of Java EE 7).
1.1	August, 2017	Addition of Configuration 1.0.
1.2	September, 2017	Addition of Health Check 1.0, Metrics 1.0, Fault Tolerance 1.0, and JWT Propagation 1.0. Updating to Configuration 1.1.

VERSION	DATE	CHANGES
1.3	January, 2018	Addition of Open Tracing 1.0, Open API 1.0, and REST Client 1.0. Updating to Metrics 1.1 and Configuration 1.2.
1.4	June, 2018	Updating to Open Tracing 1.1, REST Client 1.1, Fault Tolerance 1.1, JWT Propagation 1.1, and Configuration 1.3.
2.0	June, 2018	Updating to CDI 2.0, JAX-RS 2.1, and JSON-P 1.1 as well as adding JSON-B 1.0 (parts of Java EE 8).
2.1	October, 2018	Updating to Open Tracing 1.2.
2.2.	February, 2019	Updating to Open Tracing 1.3, Open API 1.1., REST client 1.2, and Fault Tolerance 2.0.

Most Recent Developments

As the version increment indicates, MicroProfile 2.2 is mostly a minor update of interest to existing adopters. Below are some significant changes:

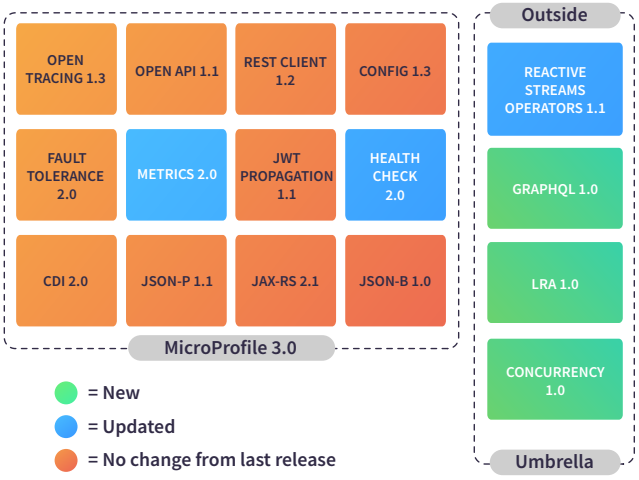
- Open Tracing 1.3 improves integration with MicroProfile Rest Clients.
- Open API 1.1 adds support for the JAX-RS 2.1 PATCH method and improves integration with MicroProfile Rest Clients.
- Rest Client 1.2 improves support for HTTP headers, especially via the new @ClientHeaderParam annotation.
- Fault Tolerance 2.0 was upgraded to CDI 2.0. In addition, the @Asynchronous annotation now supports CompletionStage.

Aside from the umbrella release, Reactive Streams Operators 1.0 was released as a standalone feature. This is a lower level SPI-style technology that will likely be incorporated into other parts of MicroProfile going forward.

A beta release of the MicroProfile Starter was just made available. This is an online project generation tool that makes it a lot easier to get started with MicroProfile projects.

Roadmap

The likely contents of MicroProfile 3.0 are shown in the graphic below. It is due to be released in June 2019. It should contain major updates to Metrics as well as Health Check. Outside of the umbrella release, Reactive Streams Operators will have a minor release when GraphQL, Long Running Actions (LRA), and MicroProfile Concurrency are projected to have their initial release. Once these APIs mature, it is likely they will be included in the umbrella release.



The topics currently actively under discussion in the MicroProfile forums include the following.

- Long Running Actions (LRA)
- Reactive Messaging
- GraphQL
- Concurrency
- Reactive Relational Database Access
- Event Data
- Service Meshes

MicroProfile APIs

The following is a high-level overview of the current MicroProfile APIs with brief code examples.

OPEN API

MicroProfile Open API provides a JAX-RS binding for the Open API specification. The Open API specification is an API description format for REST. One way to think of it is that it is WSDL (Web Services Description Language) for REST. The feature is enabled by default in all MicroProfile applications and automatically generates API documentation for JAX-RS endpoints.

```
@GET
@Operation(description="Get all current
memberships")
@APIResponses({
   @APIResponse(responseCode="200",
       description="Successful, returning
memberships"),

   ...
})
public List<Membership> getAllMemberships() {
```

The Open API `@Operation` and `@APIResponses` annotations provide overrides for what MicroProfile would automatically generate by scanning JAX-RS annotations.

Open Tracing

MicroProfile Open Tracing provides a Java binding for the Open Tracing specification. Tracing the flow of a request in a distributed environment has always been challenging. The Open Tracing specification solves this problem by providing a standard for instrumenting microservices for distributed tracing in a technology agnostic manner. One way to think of the concept is that it is a stack trace for REST services. Tools that support the concept include Jaeger and Zipkin.

```
@GET
@Path("{id}")

@Traced(operationName="GetMembershipById",
value=true)
public Membership getMembership(
   @NotNull @PathParam(value="id") int id) {
```

The Open Tracing `@Traced` annotation in the code example assigns a more meaningful name to the method being traced and specifies that call values should be traced.

JWT Propagation MicroProfile JWT Propagation provides a Java binding for the JWT (JSON Web Token) specification. REST services are usually stateless, and any security state associated with a client is sent to the target service on every request in order to allow services to re-create a security context for the caller and perform both authentication and authorization checks. This is basically what OAuth2, OpenID Connect, and JWT do. It could be thought of as SAML for REST.

```
@GET
@RolesAllowed({"admin"})
public List<Membership> getAllMemberships() {
```

MicroProfile detects the presence of a JWT token in the inbound client request and translates it under the hood to an authenticated user principal. The code above checks that the principal includes the "admin" role before allowing access.

Configuration Java EE has always included the ability to configure applications. However, such configuration is mostly static after deployment. In a cloud environment, it should be possible to modify configuration values from outside an application at runtime so that the application itself does not need to be repackaged. MicroProfile Configuration makes it possible to read configuration values from various sources such as property files, system properties and environment variables. It is even possible to read configuration values from sources such as the secure configuration store of the cloud provider (such as the Azure Key Vault). The client code for MicroProfile configuration is quite simple:

```
@Inject
@ConfigProperty(name="host.name",
defaultValue="localhost")
private String hostname;
```

The `@ConfigProperty` annotation injects the "host.name" value from somewhere in the environment. If it is not found, the default value of "localhost" is injected.

Rest Client

JAX-RS provides a powerful client API, but it can be hard to use and not really type safe. Several JAX-RS implementations support the ability to take an interface definition (typically for a JAX-RS endpoint) and create a JAX-RS client proxy from it. This is very similar to how JAX-WS/SOAP clients work. MicroProfile Rest Client standardizes this capability:

```
String apiUrl = "http://localhost:9080/
movieReviewService";
MovieReviewService reviewService =
   RestClientBuilder.newBuilder()
           .baseUrl(apiUrl)
           .build(MovieReviewService.class);
Review review = new Review(3 /*stars*/, "Good
Movie.");
reviewService.submitReview(movie, review);
```

The `MovieReviewService` in the code example is a generated proxy client (from a JAX-RS server-side endpoint) that is hydrated using the `RestClientBuilder` factory.

Health Check Health checks probe the state of a computing node from another machine (such as the Azure Kubernetes Service controller) in order to facilitate things like monitoring dashboards and auto-restarts/self-healing. MicroProfile Health Check allows the publishing of health data in a standard way

using a convenient API:

```
@Health
@ApplicationScoped

public class CheckDiskSpace implements HealthCheck
{
   ...
     public HealthCheckResponse call() {
       ...
   }
}
```

The MicroProfile `@Health` annotation publishes health information using a standard format at a standard URL. The returned `HealthCheckResponse` includes a field to indicate if a service is up or down. You can also add arbitrary metadata that might be helpful to understanding service health (such as perhaps remaining free space in our case). There are simple factories in the API to easily create and populate a `HealthCheckResponse`.

Metrics

Similar to health checks, Metrics publish common statistics in a standard location and format. This information is collected and used by distributed metrics registries and consoles such as Prometheus/Kubernetes. MicroProfile Metrics publish a required base set of metrics (such as CPU and memory) and some vendor specific metrics (such as thread pool usage information). You can also add application specific metrics.

```
@GET
@Timed(name="get_all_memberships_time",
absolute=true,
   unit=MetricUnits.MICROSECONDS)
public List<Membership> getAllMemberships() {
...
@POST
@Counted(name="memberships_created", absolute=true,
monotonic=true)
public Membership createMembership(
   ...
   Membership membership){
```

In the code example, two application specific metrics are added. `@Timed` measures the average time required to get all memberships while `@Counted` measures how many new memberships were added. Other such MicroProfile Metrics annotations include `@Gauge` and `@Metered`.

Fault Tolerance

Microservices, especially running on the cloud, are inherently unreliable. MicroProfile Fault Tolerance makes dealing with this unreliability a little easier by standardizing a set of common failure-handling patterns via annotations. Without these annotations, you would wind up with convoluted if-then-else statements in your business logic. Retries, timeouts, bulkheads, fallbacks, and circuit breakers are some of the common patterns that the API supports.

```
@GET
@Path("{id}")
@CircuitBreaker(failOn=RuntimeException.class,
requestVolumeThreshold=1,
   failureRatio=1, delay=10, delayUnit=ChronoUnit.
SECONDS)
@Timeout(value=3, unit=ChronoUnit.SECONDS)
@Bulkhead(2)
public Membership getMembership(
   @NotNull @PathParam(value = "id") int id) {
```

In the code example, the `@Timeout` annotation causes the method call to fail unless it executes within three seconds. The `@Bulkhead` annotation limits concurrent requests to two. Finally, the `@CircuitBreaker` annotation specifies that the method should not be accessed if an unchecked exception occurs. Once a failure occurs, the method can be accessed again after 10 seconds. Other annotations in the API include `@Retry`, `@Fallback`, and `@Asynchronous`.

Summary

As you can see, MicroProfile fills an important gap by providing some cool features that are perhaps even useful outside of microservices. I would encourage you to start trying them out using one of the many good MicroProfile implementations. Once you have tried things out, a very useful thing to do is get engaged with the highly active and responsive MicroProfile community!

Resources

- Home page: microprofile.io
- Starter: start.microprofile.io
- Forum: groups.google.com/forum/#!forum/microprofile

REZA RAHMAN is a Principal Program Manager for Java on Azure at Microsoft. He works to make sure Java developers are first class citizens at Microsoft, and Microsoft is a first-class citizen of the Java ecosystem. Reza has long been a frequent speaker at Java User Groups and conferences worldwide including JavaOne and Devoxx. He has been working with Java EE technology since its inception, developing on almost every major application platform ranging from Tomcat to JBoss, GlassFish, WebSphere, and WebLogic. Reza has developed enterprise systems for well-known companies like eBay, Motorola, Comcast, Nokia, Prudential, Guardian Life, USAA, Independence Blue Cross, Anthem, CapitalOne, and AAA using Java EE and Spring. LinkedIn Twitter

Beyond Java 8

BY TRISHA GEE
DEVELOPER ADVOCATE AT JETBRAINS

QUICK VIEW

01. Java has new releases every six months instead of releasing every three years.

02. Most releases will only be supported for six months until the next release. Every three years a release will be a LTS (Long Term Support) release, which will be supported for at least three years.

03. Oracle now releases two different JDKs with separate licenses, one commercial and one free, with different approaches to updates and support.

04. Oracle has added all features from their JDK to OpenJDK, so now there's no functional difference. This gives you a greater choice of JDKs from a range of vendors.

If recent surveys are to be believed (including DZone readers, see page 3), most developers are still using Java 8 for the majority of their application. Before Java 8, uptake of a new version of Java was quite slow, particularly in enterprises where it can be difficult to get a new version of Java accepted for production. Java 8, with the introduction of lambda expressions and streams, was an appealing choice for many developers. The adoption of microservices, continuous delivery practices, and better automated testing also makes it easier to use a new version of a language with far less risk than in the past.

So, given this, why is it that developers are "stuck" on Java 8, despite the fact that since Java 9 we now have two releases of Java a year? Java 12 is now the most recent version, and yet few people have made the leap to 9, 10, 11 or 12.

As you might expect, the answer is "it's complicated". There have been a number of different changes since Java 8 was released that may make organizations wary of upgrading.

Six Monthly Release Cadence

Since Java 9 (released September 2017), Oracle has been releasing a new JDK every six months, in March and September every year. In a continuous delivery world, it makes sense for our language to evolve much faster than every three years. The new release cadence means that instead of getting huge, feature-packed releases dumped on us every few years, with the associated risks of such a big upgrade, we get much smaller releases on predictable dates. These releases have far fewer features of course, but there are several benefits to this model:

1. **Easier to plan for.** The predictable release cadence makes it easier not only for language developers to plan for, but also to plan our upgrades.
2. **Higher quality.** Frequent releases mean that if a feature isn't ready for this version of Java, it won't be long until the next release. This means far less pressure on language developers to rush to complete something, and therefore higher quality releases.
3. **Steady supply of new features.** Instead of three years of potential stagnation followed by a huge upgrade, we're getting regular updates of language features, garbage collector changes, and performance improvements.

The potential downside of moving to such a rapid release cadence is that many organizations simply can't keep up with upgrading every six months. This has been fully considered, not least of all because Oracle too will be impacted by this.

Long Term Support Releases

Oracle doesn't want to support every release for three (or more) years the way they used to support the previous releases. This would be a huge cost to them.

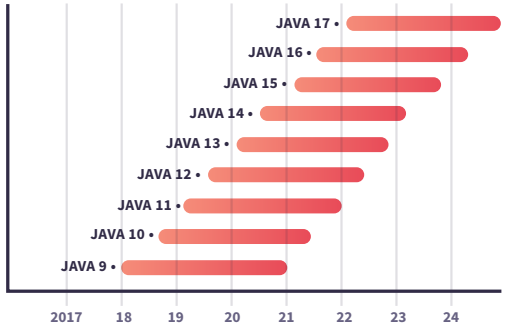


Figure 1: If Oracle supported every six-month release for three years, at some point in 2020 they'll be supporting six different versions of the language!

Instead, they've said one release every three years will be a Long Term Support release, which they will support for about three years. Java 8 was an LTS, the current LTS is Java 11, and presumably the next will be 17.

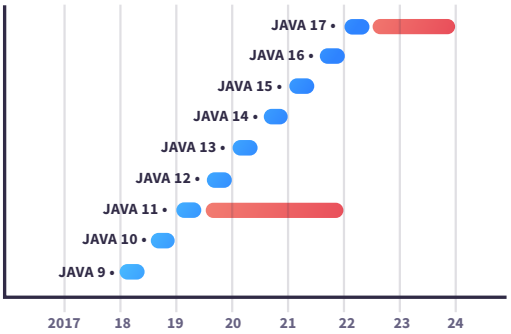


Figure 2: Oracle selects a release every three years to provide long term support and updates for, and the other releases only live for six months.

The intermediate releases, though, will not be supported once the next release is out. This means that Java 9 was replaced by Java 10, and when Java 11 was released it superseded Java 10.

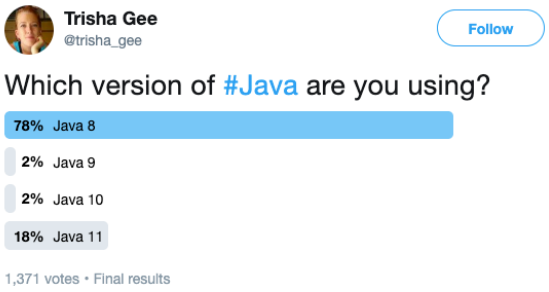


Figure 3: When developers upgrade from Java 8, they generally upgrade all the way to Java 11.

This explains this pattern of adoption — generally people will not be on Java 9 or 10. If they were using it at any stage, they should have moved on to at least Java 11.

With the combination of short-term releases that one should migrate from as soon as a new release is out, and long-term releases that are supported for at least three years, there are now potentially two main approaches to upgrading:

1. **Upgrade to every release as it comes out.** This means adopting the latest version of Java every six months. The benefit is that you'll get new features as soon as they're available, but this approach will probably only suit those who are used to upgrading their technology stack this quickly.

2. **Upgrade to just the long-term support releases.** This is a more familiar cadence for us Java developers. This gives us the downside of a Big Bang release every three years, but we have more time to evaluate the risks of such an upgrade.

There's a potential middle ground here: use the LTS release in production, but make sure your application is running against each six-month release in CI. Doing this should minimize the risk of a big bang upgrade while maintaining the required stability in production.

Oracle doesn't want to support every release for three (or more) years the way they used to support the previous releases. This would be a huge cost to them.

License Changes

There is a caveat to these long-term support releases. Oracle's

position is that if you want support and updates for three years, you should be prepared to pay for them. So, if you want to use an LTS and get updates for three years, you will need to pay Oracle for their commercial JDK.

However, Oracle recognizes that not everyone wants to do this, and that many people want to work in an open-source-friendly way, so they now have two versions of their JDK (both functionally the same) with two different licenses. They have their commercial JDK, which is free to use in development and testing but you need to pay to use it in production; and they have a zero cost OpenJDK build. The latter has an open source GPLv2+CPE licence, but will only be updated within six months of this release lifespan.

> In a continuous delivery world, it makes sense for our language to evolve much faster than every three years.

This is actually good from a competition point of view. Oracle has contributed everything from their JDK back to OpenJDK, even the features that used to be commercial features like Java Flight Recorder and Java Mission Control. So, any JDK built from OpenJDK, which is pretty much everyone's JDK, will have all the features you're used to using, and maybe even some you haven't used before.

There are plenty of other vendors offering JDKs. Many are free, some will be updated or supported for different lengths of time from the Oracle JDKs. This document lists these different options, separating those that are free to download and use from those that have a commercial model. If this all seems a little overwhelming, a good starting point is to download your OpenJDK build from AdoptOpenJDK. This JDK is compatible with the Oracle JDK, you can get builds for the major operating systems and platforms, you can choose from two different JVMs (Hotspot and OpenJ9), they have committed to providing builds for the LTS releases for at least 4 years, and you have the option to purchase commercial support.

Since this topic can be quite complex, I highly recommend reading the Java Is Still Free document produced by the Java Champions for more in depth information.

In Summary

Lots of things have changed since Java 8's release: we get releases every six months; the licensing, update, and support models have changed; and where we get our JDK from may have changed. On top of that, there were language changes, including the major changes that went into Java 9.

While all of these changes may seem daunting, their goal is to provide high quality, frequent, and predictable updates to one of the most popular languages in the world, in a way that is sustainable to those who work on and support the language.

It's worth understanding the impact these changes may have on your application and organization, because working out how to embrace them will ultimately allow you to make use of the improvements that have already gone into the language since Java 8, and will continue to evolve and improve the language every six months.

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, she believes healthy communities and sharing ideas help us to learn from mistakes and build on successes. As a Developer Advocate for JetBrains, she gets to share all the interesting things she's constantly discovering. LinkedIn Twitter

Executive Insights on the Current State of the Java Ecosystem

QUICK VIEW

01. The most important elements of the ecosystem continue to be Oracle, the Eclipse Foundation, and the members of the community.

02. IBM, Red Hat, Pivotal, Amazon, and many others have stepped up to make a significant impact on Java beyond Oracle.

03. The most significant change in the past year has been the abandonment of the JRE as many enterprises are scrambling to move to another JRE.

BY TOM SMITH
RESEARCH ANALYST, DZONE

To understand the current and future state of the Java Ecosystem, we reached out to our community for their insights. Unlike other topics like containers and security, there are far fewer people willing to share their thoughts on the current and future state of Java. This appears to be a function of its maturity relative to other technologies.

We are grateful to our three contributors who all have significant experience with Java:

- Anders Wallgren
  CTO, Electric Cloud
- Erik Costlow
  Principal Product Evangelist, Contrast Security
- Mark Little
  V.P. Middleware Engineering, Red Hat

Key Findings

1. The most important elements of the Java ecosystem are Oracle, the Eclipse Foundation, and the members of the community that ensure pervasiveness and compatibility. There is huge community adoption, low barrier to entry, nice frameworks, and toolsets. Java is available on all laptops, desktops, servers, cloud systems, and some embedded devices and mobile phones. This is invaluable since code is compatible and doesn't need to be continually recompiled or changed.

2. The most important players in the ecosystems continue to be Oracle and the community with many significant members of the community stepping up. IBM and Red Hat have made significant contributions in the language space while Amazon will have a significant impact through its default choices and the new Corretto OpenJDK distribution. Pivotal is also crucial thanks to frameworks like Spring and Spring Boot, while the Eclipse Foundation maintains Java EE.

> Having to change from the JRE hurts adoption. It's not just a question of the language, it's also the tooling and the frameworks.

3. The most significant changes in the past year is the abandonment of the JRE. It overshadows all of the good things like the consistent feature release cadence. This is delaying the adoption of Java past Java 8. There's a lot of good stuff in Java 11, but people are delaying adoption of the new features by six months to a year to move to another JRE.

4. Things Oracle and the community can do to encourage organizations to upgrade past Java 8 is to make it easy to do so. Promote Adopt OpenJDK, keep coming out with compelling improvements that make developers lives simpler and easier, and keep licensing straightforward.

Adopt OpenJDK is a one-stop shop for developers to get access to different JDK builds from different vendors. The community helps people who are having problems moving from version 8 to later versions.

Having to change from the JRE hurts adoption. It's not just a question of the language, it's also the tooling and the frameworks.

Java users need compelling reasons to move beyond Oracle to the public version. Amazon Corretto's Java 8 is now publicly supported for four years beyond Java 8. The strongest encouragement will be from the cloud providers. When Java X becomes the default in major cloud providers, people will choose Java X.

5. Java continues to be used everywhere and we can expect it to see greater adoption in the cloud and IoT. Java runtime, enterprise Java class libraries, and application servers will be tuned to better fit into new environments.

6. The most significant challenges with the Java ecosystem today are size and complexity. There's a lot going on and it can be difficult to know where to go in the community to get the help you need. The most common problem are the modularity issues above Java 8. If an application needs work to move up, people don't know what's required or it may require an obscure rework on a third-party library that no one understands. There needs to be a compelling reason to upgrade or it needs to become easier.

7. As long as large enterprises are able to evolve beyond Java 8 and Java becomes more cloud-native, it will continue to be a predominant language. Companies are unable to throw away the investment they've made in Java. Languages are hard to change. It will be interesting to see if other languages begin to use the Java Virtual Machine. There have been niche solutions for this in the JRE like JRuby, Nashorn, and Jython.

As long as large enterprises are able to evolve beyond Java 8 and Java becomes more cloud-native, it will continue to be a predominant language.

8. Developers need to examine the reasons they are considering a new language. Look at the stability of Java over the long term and the breadth and depth of the ecosystem. To deliver enterprise software you need tools, CI/CD, test suites, class libraries, and a suite of additional capabilities. It will take years for a new language to achieve the richness of Java.

Java in the enterprise is not going anywhere because of the existing code that's in place. It works, it scales, and people know how to run and operate it. What you are able to get out of the JRE with regards to the instrumentation, analysis, and operational insights during runtime is not available with other languages. The ability to get introspection while running are very interesting in the application world where there is a desire for things to run a long time.

Java is a blue collar language for developers trying to solve real business problems. As such, know the problem you are trying to solve. Java's success can be attributed to the fact it has removed distractions like memory management and certain aspects of security. When reading code you can see the problem being solved.

TOM SMITH is a Research Analyst at Devada 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. LinkedIn - Twitter

Java 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	Open Source?	Website
Adzerk	Hoplon	ClojureScript web framework	Open source	hoplon.io
Alachisoft	NCache	In-memory data grid (JCache-compliant)	Open source	alachisoft.com/ncache
Amazon Web Services	AWS ECS	Elastic container service w/Docker support	Free tier available	aws.amazon.com/ecs
AngularFaces	AngularFaces 2.1	AngularJS plus JSF	Open source	angularfaces.net
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 w/JAX-WS & JAX-RS support	Open source	cxf.apache.org
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 mgmt w/strong Ant integration	Open source	ant.apache.org/ivy

Company	Product	Product Type	Open Source?	Website
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 mgmt/ 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 mgmt)	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
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

Company	Product	Product Type	Open Source?	Website
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
Atomist	Atomist	Self-service software delivery	Free tier available	atomist.com
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
BMC	TrueSight	Infrastructure monitoring	14 days	bmc.com/it-solutions/truesight.html
Bouncy Castle	Bouncy Castle	Java & C# cryptography libraries	Open source	bouncycastle.org
CA Technologies	CA Application Performance Management	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
Cask	Cask	Data & application integration platform	Open source	cask.co
Catchpoint	Catchpoint	APM w/Java agent	14 days	catchpoint.com
CData	CData	Data integration & connectivity	Available by request	cdata.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 mgmt	Open source	chef.io/chef
Chronon Systems	DripStat APM	Java & Scala APM w/many framework integrations	Free tier available	dripstat.com/products/apm

Company	Product	Product Type	Open Source?	Website
Cloudbees	Jenkins	CI server	Open source	jenkins.io
Codeborne	Selenide	UI tests in Java (Selenium WebDriver)	Open source	selenide.org
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 versions available	sysdig.com
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 tier available	ej-technologies.com/products/jprofiler/overview.html

Company	Product	Product Type	Open Source?	Website
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 Digital	Predix	IIoT platform w/Java SDK (on Cloud Foundry)	Demo available by request	ge.com/digital/iiot-platform
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
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
The Grails Project	Grails	Groovy web framework (like Ruby on Rails)	Open source	grails.org
GridGain Systems	GridGain	In-memory data grid (Apache Ignite & enterprise mgmt, security, monitoring)	Free tier available	gridgain.com
H2	H2	Java SQL database engine	Open source	h2database.com
Haulmont	CUBA Platform	Java rapid enterprise app development framework	Open source	cuba-platform.com
Hazelcast	Hazelcast IMDG	Distibuted in-memory data grid (w/ JCache implementation)	Open source	hazelcast.org
HyperGrid	HyperForm	Container composition platform	30 days	hypergrid.com
IBM	IBM Cloud	PaaS w/extensive Java support	Free tier available	ibm.com/cloud
IBM	WebSphere Application Server	Java application server	60 days	ibm.com/cloud/websphere-application-platform

Company	Product	Product Type	Open Source?	Website
IceSoft	IceFaces	JSF framework	Open source	icesoft.org/java/projects/ICEfaces/overview.jsf
Informatica	Informatica	Data integration & mgmt	30 days	informatica.com
Integral GmbH	FusionReactor	JVM APM w/production debugging & crash protection	Available by request	fusion-reactor.com
Isomorphic Software	Smart GWT	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	Open-source version available	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	Available by request	jclarity.com/censum
jClarity	Illuminate	Java-focused APM w/machine learning & autosummarization	Available by request	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	Open-source version available	jetbrains.com/idea
JFrog	Artifactory	Binary/artifact repository manager	30 days for on-prem, 14 days for cloud	jfrog.com/artifactory
JFrog	Bintray	Package hosting & distribution infrastructure	Open-source version available	bintray.com

Company	Product	Product Type	Open Source?	Website
Jinfonet	JReport	Reporting, dashboard, analytics, & BI for Java	Available by request	jinfonet.com
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	Open source	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	30 days	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)	Demo available by request	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
The Linux Foundation	Kubernetes	Container orchestration	Open source	kubernetes.io
Machinery for Change	CP3O	JDBC connection & statement pooling	Open source	mchange.com/projects/c3p0
ManageCat	ManageCat	Manage, monitor, & troubleshoot Apache Tomcat	Available by request	managecat.com
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	Open Source?	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	Free tiers available	nastel.com
Netflix	Hystrix	Latency & 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
The Netty Project	Netty	Event-driven, non-blocking JVM framework for protocol clients & servers	Open source	netty.io
New Relic	New Relic	APM w/Java agent	Demo available by request	newrelic.com
NGINX	NGINX	Web server, load balancer, reverse proxy	Open-source version available	nginx.com
Ninja Framework	Ninja Framework	Full-stack web framework for Java	Open source	ninjaframework.org
Nuxeo	Nuxeo Platform	Structured & rich content mgmt platform	Demo available by request	nuxeo.com
Object Refinery Limited	JFreeChart	Java charting library	Open source	jfree.org/jfreechart
Okta	Okta Identity Cloud	Technology integrations	30 days	okta.com
OmniFaces	OmniFaces	JSF utility library	Open source	omnifaces.org

Company	Product	Product Type	Open Source?	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
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 19c	Relational DBMS	N/A	oracle.com/database/technologies
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	Demo available by request	outsystems.com
OverOps	OverOps	JVM agent for production debugging	Available by request	overops.com
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

Company	Product	Product Type	Open Source?	Website
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/software/mysql-tools
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	spring.io/projects/spring-boot
Pivotal	Spring Cloud	Distributed systems framework (declarative, opinionated)	Open source	spring.io/projects/spring-cloud
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	spring.io/projects/spring-framework
Pivotal	Spring MVC	Server-side web framework	Open source	docs.spring.io/spring/docs/current/spring-framework-reference/web.html
Plumbr	Plumbr	Memory leak detection, GC analysis, thread & query monitoring	14 days	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	30-90 days	developer.thingworx.com
PubNub	PubNub	Real-time mobile, web, & IoT APIs	Free tier available	pubnub.com
Puppet Labs	Puppet	Infrastructure automation/ configuration mgmt	Open-source versions available	puppet.com
Push Technology	Diffusion	Real-time messaging (web, mobile, IoT)	Free tier available	pushtechnology.com
Qoppa Software	Qoppa PDF Studio	PDF manipulation from Java	Free tier available	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

Company	Product	Product Type	Open Source?	Website
Red Hat	Ansible	Deployment automation & configuration mgmt	Open source	ansible.com
Red Hat	Codenvy IDE	SaaS IDE w/dev workspace isolation	Free tiers available	codenvy.com
Red Hat	Drools	Business rules mgmt 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
Red Hat	Thorntail	Packaging & running Java EE aplications	Open source	thorntail.io
Red Hat	WildFly	Java application server	Open source	wildfly.org
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	Available by request	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	14 days	saucelabs.com/open-source
Scalatra Team	Scalatra	Scala web microframework	Open source	scalatra.org

Company	Product	Product Type	Open Source?	Website
Slenium	Selenium	Browser automation w/Junit & TestNG integration	Open source	seleniumhq.org
Software AG	EHCache	JCache implementation	Open source	ehcache.org
Software AG	Terracotta BigMemory Max	In-memory data grid w/Ehcache (JCache implementation)	Available by request	terracotta.org/bigmemory-and-web-sessions
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 & specification framework for Java & Groovy	Open source	spockframework.org
Square	Dagger	Dependency injector for Android & Java	Open source	github.com/google/dagger
Synopsys	Black Duck Platform	Security & open-source scanning & mgmt (w/container support)	Open source	blackducksoftware.com
Teradata	Teradata	Data warehousing, analytics, lake, SQL on Hadoop & Cassandra, big data appliances, R integration, workload mgmt	Open-source version available	teradata.com
TestNG	TestNG	Java unit testing framework (JUnit-inspired)	Open source	testng.org/doc
Thinking Software, Inc.	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
Trend Micro	Immunio	Runtime application self-protection w/Java support	Free tier available	immun.io
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 & Finagle	Open source	twitter.github.io/finatra

Company	Product	Product Type	Open Source?	Website
Vaadin	Vaadin	Server-side Java -> HTML5	Open-source version available	vaadin.com
Vert.x	Vert.x	Event-driven, non-blocking JVM framework	Open source	vertx.io
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
Whitesource	Whitesource	OSS component security	Open-source version available	whitesourcesoftware.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 mgmt	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
Yonita	Yonita	Automated detection of code defects	N/A	yonita.com
ZeroTurnaround	JRebel	Class hot-loading (in running JVM)	Available by request	zeroturnaround.com/software/jrebel
ZeroTurnaround	XRebel	Java web app profiler	10 days	zeroturnaround.com/software/xrebel
Zkoss	ZK Framework	Enterprise Java web framework	Open source	zkoss.org
Zoho	Site24x7	Website, server, APM	30 days	site24x7.com

INTRODUCING THE

Java Zone

Java remains the most popular programming language even as it continuously evolves. From updated APIs to new JDK 11 and JDK 12 features, there's plenty to unpack in the current Java ecosystem.

Keep a pulse on the industry with topics such as:

- New Java APIs
- Epsilon and garbage collection
- Java EE and Jakarta EE
- Spring framework
- Other JVM language

Visit the Zone

TUTORIALS

CASE STUDIES

BEST PRACTICES

CODE SNIPPETS