DZone

THE 2018 DZONE GUIDE TO

Java

FEATURES, IMPROVEMENTS, & UPDATES

VOLUME IV

RESEARCH PARTNER SPOTLIGHT

ORACLE®

Key Research Findings

BY G. RYAN SPAIN
PRODUCTION COORDINATOR, DZONE

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

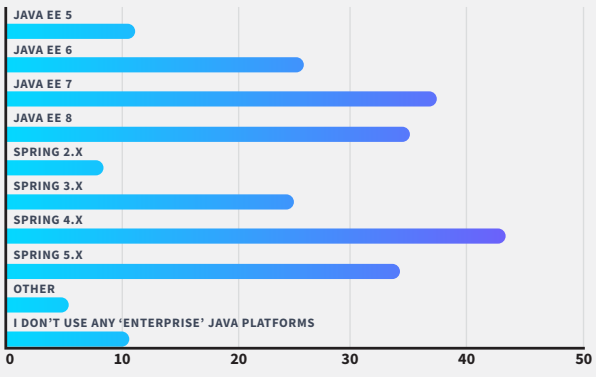
- 41% of respondents identify as developers or engineers; 20% identify as developer team leads; and 17% identify as architects.
- The average respondent has 13.7 years of experience as an IT professional. 58% of respondents have 10 years of experience or more; 21% have 20 years or more.
- 41% of respondents work at companies headquartered in Europe; 31% work at companies with HQs in North America.
- 21% of respondents work at organizations with more than 10,000 employees; 18% at organizations between 1,000 and 10,000 employees; and 23% at organizations between 100 and 999 employees.
- 87% develop web applications or services; 53% develop enterprise business apps; and 22% develop native mobile applications.

JAVA -VERSION

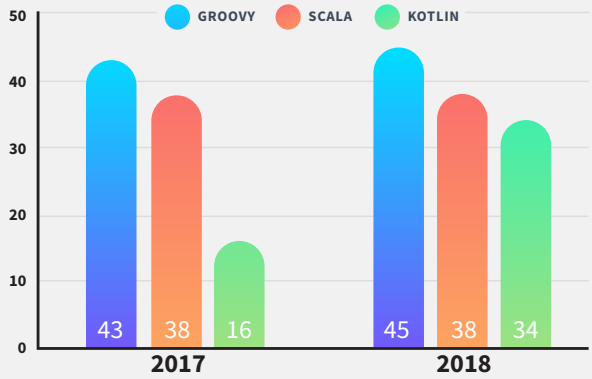
Since DZone's 2017 Guide to Java Development and Evolution was released, Java has continued to… evolve. Between the publication of that guide and the guide you're reading now, Java 9 and Java 10 have been released, and public updates for Java 9 have ended. By the time DZone's next Java Guide is released,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

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

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



GRAPH 02. JVM language usage



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

KOTLIN, KOTLIN, KOTLIN!!!

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

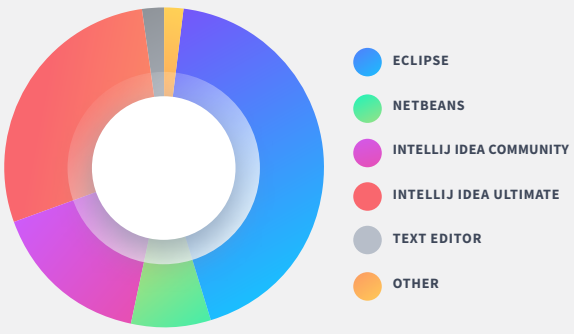
THE ART OF JAVA IS LARGELY THE ART OF PERSISTENCE

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

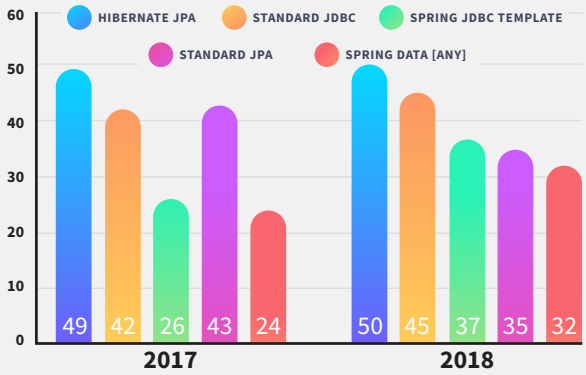
FRONT-END FREE-FOR-ALL

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

GRAPH 03. Where do you primarily write Java code?



GRAPH 04. Java data persistence tool usage



JAVA IS IN TENTS

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

JAVA 8

- Saw the introduction of many features related to functional programming into the Java language, such as lambda expressions, functional interfaces, and the Stream API.
- Introduced the Nashorn engine to enable running dynamic JavaScript code natively on the JVM.
- Included an overhaul for JavaFX, such as a new theme (Modena) and making it available for ARM platforms.

JAVA 9

- Modularized the Java Virtual Machine with Project Jigsaw (officially known as the Java Platform Module System).
- Brought a REPL to Java in the form of JShell – an interactive command-line interface.
- Improved and extended the JAR file format to allow for multiple Java release-specific versions of class files to

JAVA 10

- The first release in a new six-month cycle, Java 10 has been a relatively minor release leading up to Java 11.
- Brought local variable type inference to Java with the var keyword.
- Enabled Java's JIT compiler, Graal, to be used as an experimental compiler on the Linux/x64 platform.

ATTITUDES

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

JAKARTA EE

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

SPRING FRAMEWORKS

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

GARBAGE COLLECTION

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

* *Features of all Java Versions*

COPYRIGHT 2018 DZONE.COM

LOOKING AT JDK 10 AND JDK 11

WRITTEN BY DUSTIN MARX

JDK 10

SUMMARY JAVADOC TAG

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

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

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

CODE CONTINUED ON NEXT COLUMN

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

ACCESSING A JAVA APP'S PROCESS ID FROM JAVA

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

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

FUTURETASK GETS A TOSTRING()

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

JDK 11

FIVE NEW JEPS TARGETED FOR JDK 11

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

1. JEP 324

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

2. JEP 327

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

3. JEP 328

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

4. JEP 329

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

5. JEP 330

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

NEW METHODS ON JAVA STRINGS

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

NEW METHODS ON JAVA FILES

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

NEW PREDICATE NEGATION

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

Shaping the Future of Java...Faster

PRODUCT
Java SE

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

TWITTER
@OpenJDK

WEBSITE
openjdk.java.net

ORACLE

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

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

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

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

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

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

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

- Portola: Java's characteristics make it ideal for container deployment, such as Docker, and the project goal to provide a port of the JDK to the Alpine Linux distribution.
- ZGC: The project goal is to provide a scalable, low latency garbage collector that does not require tuning.
- Valhalla: Java is very good at optimizing code and the project goal is to also optimize data.
- Panama: With opportunities in big data and machine learning, the project goal is to allow access to low-level hardware functionality through normal Java code.
- Amber: Coding productivity is key, and the project goal is to make Java a much less verbose and approachable language for developers.

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

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

Join the World's Largest Developer Community

Download the latest software, tools, and developer templates

Get exclusive access to hands-on trainings and workshops

Grow your network with the Developer Champion and Oracle ACE Programs

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

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

@OracleDevs **facebook.com/OracleDevs**

ORACLE®

Copyright © 2018, Oracle and/or its affiliates. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates.