What keeps Java vibrant?

“We’ve always been guided by a clear set of values,” says Reinhold, “and we’ve taken the notion of stewardship very seriously.”

“We don’t add stuff to Java based on what’s cool this year,” he adds, or because language x or language y has a specific function. “We take a deep amount of care when we add things to the platform, when we make changes, and when we occasionally, rarely, remove things.”

To Reinhold, the Java language and platform emphasize readable code. “Readability is a key value. The other key value, I would say, is compatibility. We take compatibility seriously, even though we do occasionally remove things, as we have recently.”
Not much has been removed, Reinhold adds: "There are many, many dusty JAR files out there, created on Java 1.0, Java 1.2, and Java 1.4, that work just fine today. That's not something you can say about other platforms."

Another big factor in the language's longevity has been the ecosystem, the community as a whole, Reinhold points out. "We've been able to maintain an atmosphere of support and collegiality. Plus, we have all the user groups and conferences around the world, focused on Java and related technologies."

Over the years, the reasons for Java's popularity might have changed, observes Pressler. "These days, Java is recognized as a serious platform, for serious software. The platform is dependable, and it offers an unparalleled combination of performance, productivity, and observability—by which I mean monitoring and profiling."

"The stewardship and the values that we bring to evolving the Java platform are augmented with an unusual blend of theory and practice," says Goetz. "Where the platform meets the programming model, which is what most developers see every day, we have focused on evolving the platform, so it stays relevant to the problems people want to solve."

Goetz continues, "At the same time, down in the engine room, the specification and implementation of the VM, the JIT [just-in-time compiler], the garbage collector, the language specification—we take an immense amount of care with evolving these technologies. And, as a result, we are able to continue to evolve the specification and implementation 25 years later."

Vidstedt points to stability: "The Java platform has a really stable interface. The surface is very well defined, which means that, again, those old dusty JAR files from the 1990s still mostly run. This is complex, and the complexity can be challenging, as an implementer, to work with. We are able to spend that time solving these complex problems, so that users and developers don't really have to care about it as much."

"The well-defined, stable interface leaves a lot of room for innovation beneath the surface," adds Vidstedt. "JVM technology is a cool technology. There are a lot of different challenging areas, and we have a really healthy community around this. A lot of people are investing in Java, all the way from companies to academia—and developers help us figure out what to do next."
How does the Java team prioritize which Java language features to implement?

“The question suggests that our methodology is to start with a big, huge, long list of features, and we just prioritize everything in it,” laughs Reinhold, saying that’s not how the Java Platform Group works. “What we’re always asking is, ‘Should we do inside the core of the platform with the language, the JVM, the libraries, that which cannot be done outside of the platform?’”

Reinhold explains that machine learning is a hot area, and this has led to numerous suggestions, such as to implement the bfloat16 floating-point format or a TensorFlow API. “These are all well-meaning suggestions—but implementing those would be the wrong thing. Anybody should be able to create a TensorFlow API, and anybody should be able to define a new floating point type. But today, that’s not possible.”

That’s why, Reinhold says, the Java Platform Group’s emphasis is on making Java more extensible. “For example, rather than add bfloat16 to our little stable of unfortunate primitive types, we’re working in Project Valhalla to make it possible for anyone to define their own effectively new primitive types.”

“The principle of doing, in the platform, what just absolutely cannot be done outside it, is a good guiding one,” Reinhold adds. “That’s what we constantly look for.”

Libraries are key, says Goetz. “When we’re designing language features, we ask, ‘How can language features enable people to write more powerful libraries?’ Library changes scale much better than language changes. Language changes take a long time and affect everyone. However, if we can enable the ecosystem to solve in libraries the problem developers want solved, and to compete for who has the best TensorFlow library or JSON parsing library, then everyone wins.”

Should Java programmers give up setters and model their domain objects immutably, or with records only, from here on out?

“The either/or that’s implicit in the question, suggesting there’s only one right way to model data in your program, is a little bit misleading,” says Goetz. “Our colleague Alex Buckley, specification lead for the Java language and JVM, likes to say that one of the surprising things is that Java managed to get all the defaults wrong and still succeed.”

That’s why, Goetz says, in Java, the default for mutability is wrong. The default for accessibility is wrong. The default for whether a class is extensible is wrong. “Yet, somehow, we managed to survive these mistakes.”

“The JavaBeans naming convention is a little bit of an unfortunate story,” he continues. “It’s a terrible
for the Java Virtual Machine

Ron Pressler, Consulting Member, Java Technical Staff

convention for general-purpose development. It was a fairly well-targeted convention for naming visual components in a visual editor, where you were dragging and dropping buttons and labels and things like that onto a canvas, and you had property sheets that you could edit about foreground color and fonts and what have you, which could be discovered reflectively.”

Goetz explains that the naming convention was great way building those UI components—which turns out to not have been a key Java use-case. “And yet, somehow, we continued with that naming convention, even though it’s such a poor match for most business domain objects.”

“Because of these conventions, mutability is greatly overused and overrated,” says Goetz. “Immutability is a tool for managing complexity, because it lets us focus our attention on the parts of the project that actually have to be complex.”

The answer is records. “Records is a language feature that recognizes the importance of immutability,” says Goetz. “Records makes it easier to model data as data, and together sealed types and pattern matching are moving Java towards more of a data-oriented model of programming.”

Goetz continues, “Programs are getting smaller, as we use more smaller services, microservices, and functions as a service. More code is closer to the boundary and developers will have to deal with data that comes in not as a Java object, but as some untied string of XML or JSON or whatever. We want to model that data as ordinary data. The Java language is moving in a direction that makes it easier to do that.”

What would you say to developers who are stuck on Java 8 or older versions, to get them to jump in and swim in the pool to enjoy current Java releases?

“Those developers are probably losing a significant amount of money that using a modern JDK could save them,” says Pressler. “Improvements to memory footprint, such as compact strings; improvements to performance, such as better compiler optimizations and faster startup; improvements to both footprint and performance, due to massive changes in the G1 garbage collector and the introduction of ZGC, a garbage collector that gives you pauses of one or two milliseconds; as well as improvements to the Oracle Java Flight Recorder, which can help optimize application code. All of these translate to significantly less hardware that’s required to serve the same workload. In many if not most cases, I think the savings in hardware hosting would offset the migration costs very quickly.”

Also, Pressler adds, “Who wants to be below the security baseline to begin with? Do you really want to put your application or your enterprise
at risk by using old software? It’s shocking, sometimes, when you come across just how frequently that is the situation, because organizations did not know. So, now you know.

With Project Loom, is it worth keeping reactive programming? If so, why?

“Reactive programming emerged to allow servers to better utilize their hardware for better throughput,” says Pressler. “This was one of the best answers to the problem of high-throughput servers that we could do without changing the JDK. And in some cases, for application requirements such as that, reactive programming was the only choice.”

That said, Pressler admits, despite being necessary in many situations, reactive programming clashes with the design of the Java platform. “You lose the troubleshooting information contained in exceptions. You lose the Oracle Java Flight Recorder performance profiles and the debugger interface. Those tools assume that the context of a logical application transaction is tied to the state of a thread.”

Unfortunately, Pressler adds, that assumption runs deep in the JVM, the tooling, and the Java language. “And reactive programming sort of breaks everything.”

“When high throughput is required, reactive programming might be the best choice, and in some cases, you have no choice,” Pressler points out. “However, Project Loom tries to solve the same problem, in a way that’s harmonious with the design of the Java platform and its tools. Also, Loom makes integration with legacy code easier.”

Because Project Loom changes the Java platform itself, behind the scenes, programmers can continue to use their familiar APIs. “My guess is that when Loom arrives, many developers will prefer getting better throughput by using virtual threads,” predicts Pressler. “Of course, everyone who prefers the reactive style is free to keep it. It’s also possible that reactive libraries might even find ways to use virtual threads to reduce that mismatch between the reactive programming style and the Java platform.”

What are some of the next big things that Java developers should know about?

Reinhold says that after the releases of Project Valhalla and Project Panama, he is looking forward to type classes. “It would also be good to get back to some of the ideas that Ron [Pressler] was exploring early on in Loom, particularly around tail calls and continuations. Continuations, in particular, are an extremely powerful feature.”

“It’s not clear to me that we would ever want to expose tail calls and continuations in the platform for general use,” Reinhold adds, “but there may be ways we could use them inside the platform, in some fairly interesting ways.”

Goetz points out that developers need to recognize that projects such as Valhalla are not unitary projects. “It’s not like Valhalla is going to make one delivery, and then it’s going to be done. There are multiple things coming out of each of these projects—and each of these will enable more interesting things in the future.”
For example, Goetz says, you could think about Valhalla in a couple of different ways. "You could think about Valhalla as flatter, denser data types, which is largely a performance-oriented view of looking at it, and that's true. You could also look at Valhalla as unifying the type system so that primitive types and objects are unified in a sensible model. This affects how we develop code."

"Everybody has had the unpleasant experience of having to hand-specialize some code, where you've written a nice, generic implementation of something, but now you have to do a special version for int arrays or long arrays or whatever," Goetz adds. "This kind of complexity ripples and can particularly affect libraries. By unifying the type system, we will enable a lot more interesting libraries to be written."

Goetz laughs: "We will never run out of things to do. There will always be things that are just outside the reach of what we're able to do. That said, projects such as Valhalla are a good example of both giving us more to work with and giving us new features, while leaving things with a smoother surface to build on top of in the future."

"I'll close by saying I don't know what the next big thing is, but I can hope," says Pressler. "I'm looking forward to working on tail call optimization. Something that could be modernized is the gestalt experience of using the various tools. The developer experience is something I hope we can improve in the near future."

[This article was excerpted and adapted from the panel "Ask the Java Architects" at Oracle Developer Live. —Ed.]

Dig deeper

- Is it time for operator overloading in Java?
- Records come to Java
- The role of preview features in Java 14, Java 15, Java 16, and beyond
- Project Amber
- Project Loom
- Project Panama
- Project Valhalla

Alan Zeichick

Alan Zeichick is editor in chief of Java Magazine and editor at large of Oracle's Content Central group. A former mainframe software developer and technology analyst, Alan has previously been the editor of AI Expert, Network Magazine, Software Development Times, Eclipse Review, and Software Test & Performance. Follow him on Twitter @zeichick.

Share this Page

Facebook
Twitter
LinkedIn
Email