Venkat Subramaniam: Java is changing in a responsible manner

When Java shows up at a party, it always brings a delicious dessert.

by Alan Zeichick

Dr. Venkat Subramaniam, instructional professor, the University of Houston

September 15, 2020

As part of the Moved by Java initiative, Java Magazine caught up with Dr. Venkat Subramaniam a few days before the official Java 15 launch. A well-known Java Champion—and unabashed enthusiast for all aspects of software development—Subramaniam is an award-winning author, founder of Agile Developer Inc., and an instructional professor at the University of Houston. His most recently book is Programming Kotlin: Create Elegant, Expressive, and Performant JVM and Android Applications, published in 2019 by Pragmatic Bookshelf.

Java Magazine: Java 15 is coming out, and Java itself is 25 years old. Would you describe Java as mature…or as vibrant?
Subramaniam: Languages show their age when they come up to 10 years, let alone 25 years. But in one way, Java hasn't been the same language that whole 25-year period. There have been different generations of developers involved—and I mean this with all respect—but the people who spearheaded the language in the beginning are not the people doing it today.

Java is under new hands, it's under new thoughts, and it's under a new generation as well.

Think about C++ and COBOL: Those were destination languages. C++ was C++. COBOL was COBOL. But Java is not just a language: Java is an ecosystem. We have Java the language, and we have the JDK, and we have the JVM. That is a phenomenally rich playground for innovation and experimentation.

Meanwhile it supports languages such as Kotlin and Scala and Groovy and Clojure, and all of those languages bring innovation on their own. They serve as a proof of concept to what can happen on the JVM.

The excitement for me is that Java is taking in those other ideas and experimentation, and in a very responsible way, not in an uncontrolled manner.

You know, I'll be honest about that. I am the first person who hates complexity. When I work with teams, I always shake my head and ask, "Why are we doing this? This is unnecessary." Yet when I look at innovation in Java, I have not once said to myself, "This is going overboard." The growth is at a reasonable pace, not an uncontrolled pace. The new features being added are prudent. They are not driven by infatuation, but by pragmatism. Java's growth is heading in the right direction at reasonable pace.

Java Magazine: Talk to me about that experimentation—those new ideas that have reinvigorated the platform.

Subramaniam: The world around us is changing. You know, the first thing that got me excited about Java was applets. We did not even know that Java would thrive on the server side; that came much later.

But today we are in a very different world. Back then, we did not have big data, we didn't have smart devices, we didn't have functions as a service, and we didn't have microservices. If Java didn't adapt to the new world, it would have gone extinct.

I started with Java fairly early on, and it's absolutely phenomenal and refreshing to know that I am now programming with the next generation of programmers. The desires and needs and expectations of the next generation are not the same as those of my generation.

Java has to cater to the next generation of programmers. This is a perfect storm for the language: On one hand, Java is popular
today. On the other hand, Java must stay relevant to the changing business environment, changing technology, and changing user base. And we are going to make this possible.

After 25 years, Java is not the same Java. It's a different Java, and that's what excites me about it.

**Java Magazine:** You mentioned that other JVM languages bring ideas into Java, and I'm sure that other languages and platforms have as well. Doesn't that open Java up to accusations of being derivative, rather than innovative?

**Subramaniam:** Many times I hear developers say, “Oh, Java copied this from xyz.” That's a very wrong way to think about it. When I look at Java, I see innovation. Here's the phrase I always use: “Sometimes, yes, Java came late to the party. But it came with an amazing dessert.” And that makes it worth it all the way.

Java was not the first language to bring in lambdas. But when lambdas came to Java, the innovation was *invokedynamic*, which removed the overhead of lambda implementation in the JVM.

Here's another phrase: “The rising tide lifts all boats.” Running in the JVM with *invokedynamic*, Scala is better. Kotlin is better. Groovy is better. All these languages are better on the JVM, thanks to the implementation of lambdas in the JVM.

And in the same token, I've used heredocs [multiline strings, similar to *text blocks*—*Ed.*] in multiple languages. Honestly, I love many languages, but here's what I don't like about multiline strings in some languages. When I have multiline strings within a function showing different indentation, I've got to spend effort to tell the program that I don't mean *this* indentation to be used, because it's an artificial interpretation—not an intended indentation.

[By contrast] Java has a smart algorithm to decide on the indentation for text blocks. And I was thrilled because that is innovation. Java didn't say text blocks, “Others do it; let's do it too.” Instead, Java said, “Others do text blocks, but let's make this better for programmers.”

**Java Magazine:** Do you have other examples of Java's vibrancy?

**Subramaniam:** There are amazing things happening in Java today, and there are amazing things I'm expecting, and I can't wait for them.

For example, *switch expressions* is amazing. It gives me the ability to avoid side effects when I'm writing *switch* statements because it's a *switch expression*. This capability leaps further on and eventually do pattern matching.

What drew me into languages like Erlang and Scala was the pattern-matching syntax. I remember sitting at my computer and
saying, “This pattern matching is so cool in Erlang and Scala. Why can't they do that in Java?” Well, now, it's coming to Java.

You know, a few years ago, I would ask the question, “Do you work for the compiler, or does the compiler work for you?”

**Java Magazine: [laughing]**

**Subramaniam:** That’s really how the old Java was! The old Java compiler would come to you and say, “Hey, how's it going?” And you’d say, “I'm working!” The old Java compiler would say, “Go on; I'll be back after lunch and check on you,” and you’d be left sitting there slogging away.

Today, the Java compiler works for you. Take smart casting [where the compiler tracks conditions inside an if expression — *Ed.*]. Smart casting checks whether a reference is of the appropriate type. Now you can intelligently use references in the code without having to perform cast after cast after cast, so the language feels like it’s in sync with you. That to me is huge.

Moving forward, I am blown away by records. Many people say records are great because records avoid boilerplate code. Sure; that’s great. But I’m really excited looking forward to the next few years, when records become part of pattern matching. That can lead to destructuring.

To me, this is like, whoa: In the world where we live, we do message processing. We’re getting messages of different types, and a message is a type of record. We'll be extracting data out of records with the pattern-matching syntax and doing destructuring. I’m really looking forward to that day.

Similarly, look at the sealed classes feature, which is part of Java 15. Sealed classes bring in the computer science concept of algebraic data types, where on one extreme you have abstract base classes and you can have any number of derived classes. The other extreme is final classes, where you cannot have any derived classes. Sealed classes represent a middle ground: Here’s my class, and here are the *bona fide* children for that class.

Sealed classes will shine if you use a switch statement, do pattern matching, and provide the combination of classes for the sealed classes. We won’t need to provide a default path because the compiler is able to determine that all paths and all options are covered. This makes the code a lot more reliable. If we ever add a new subclass, the compiler will immediately give us an error. That makes the code becomes a lot more maintainable as well.

Many of these new Java features are great by themselves. In combination with other features, though, they are really changing the way programmers work. I’m going to bet my money on this: In five years from now, Java will look very different than it looked the past 20 years. You are still going to be leveraging the power
of what it is sitting on, and you are also going to enjoy the capabilities of what’s been added to the language. I couldn’t be more excited about where Java is heading for that reason.

Dig deeper

- “Inside Java 15: Fourteen JEPs in five buckets”
- Java’s 25 years of ongoing innovation
- “Records come to Java”
- Sealed classes
- “Inside Java 13’s switch Expressions and Reimplemented Socket API”
- Text blocks
- “Understanding Java Method Invocation with invokedynamic”
- “Our world, moved by Java”

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.