From ‘Java Sucks’ to ‘Java. . . Eh, Not Bad’

How Vert.x & Java 21 Made Me Stop Complaining

Thomas Gebert

Who Am I?

- Software Engineer in New York City.

Who Am I?

- Software Engineer in New York City.
- There is nothing else interesting about me.

Java.

Java.

- If you are at this conference, you probably have an opinion of Java

Java.

- If you are at this conference, you probably have an opinion of Java
- Likely very negative.

Java.

- If you are at this conference, you probably have an opinion of Java
- Likely very negative.

Why the Java Hate?

Why the Java Hate?

- Java is bloated and verbose.

Why the Java Hate?

- Java is bloated and verbose.
- Encourages bad practices.

Why the Java Hate?

- Java is bloated and verbose.
- Encourages bad practices.
- Java progammers. . .

Examples of (Historically) Bad Things In Java.

Examples of (Historically) Bad Things In Java.

- IO is blocking by default

Examples of (Historically) Bad Things In Java.

- IO is blocking by default
- `synchronized` is evil.

Examples of (Historically) Bad Things In Java.

- IO is blocking by default
- `synchronized` is evil.
- Replace with concrete examples.

So Why Would We Want To Even Use Java?

So Why Would We Want To Even Use Java?

- A metric ton of well-tested and supported libraries and guides online.

So Why Would We Want To Even Use Java?

- A metric ton of well-tested and supported libraries and guides online.
- Relatively portable, even still.

So Why Would We Want To Even Use Java?

- A metric ton of well-tested and supported libraries and guides online.
- Relatively portable, even still.
- Lots of great tooling around the language in the form of IDEs and benchmarking tools available.

So Why Would We Want To Even Use Java?

- A metric ton of well-tested and supported libraries and guides online.
- Relatively portable, even still.
- Lots of great tooling around the language in the form of IDEs and benchmarking tools available.
- (Can be) fast.

Why not Kotlin? Or Clojure?

Why not Kotlin? Or Clojure?

- You *should* use Clojure if you can!

Why not Kotlin? Or Clojure?

- You *should* use Clojure if you can!
- A lot of companies still have tens of thousands of lines of Java that already exist.

Why not Kotlin? Or Clojure?

- You *should* use Clojure if you can!
- A lot of companies still have tens of thousands of lines of Java that already exist.
- Many companies will find it infeasible to migrate to a better language, and would rather spend infinitely more money hiring dozens of engineers to write a million incrementatal patches to a Java codebase.

Why not Kotlin? Or Clojure?

- You *should* use Clojure if you can!
- A lot of companies still have tens of thousands of lines of Java that already exist.
- Many companies will find it infeasible to migrate to a better language, and would rather spend infinitely more money hiring dozens of engineers to write a million incrementatal patches to a Java codebase.
- Many of us are stuck in this hell.

Modern Java

Modern Java

- In 2024 I took a job doing Java full-time.

Modern Java

- In 2024 I took a job doing Java full-time.
- They were unreceptive to my plees to use Clojure, no matter how much I complained.

Modern Java

- In 2024 I took a job doing Java full-time.
- They were unreceptive to my plees to use Clojure, no matter how much I complained.
- Eventually, I realized that I wasn't going to win this fight and instead I should at least figure out what Java 21 had to offer.

Modern Java

- In 2024 I took a job doing Java full-time.
- They were unreceptive to my plees to use Clojure, no matter how much I complained.
- Eventually, I realized that I wasn't going to win this fight and instead I should at least figure out what Java 21 had to offer.
- Much to my astonishment, *I actually enjoyed it!*

Modern Java

- In 2024 I took a job doing Java full-time.
- They were unreceptive to my plees to use Clojure, no matter how much I complained.
- Eventually, I realized that I wasn't going to win this fight and instead I should at least figure out what Java 21 had to offer.
- Much to my astonishment, *I actually enjoyed it!*

What Changed?

What Changed?

- Since Java 8 and Java 11, there has been a much higher emphasis on functional programming concepts and updated syntax to facilitate it.

What Changed?

- Since Java 8 and Java 11, there has been a much higher emphasis on functional programming concepts and updated syntax to facilitate it.
- Java programmers have *finally* joined the 21st century and will *occasionally* use non-blocking IO.

What Changed?

- Since Java 8 and Java 11, there has been a much higher emphasis on functional programming concepts and updated syntax to facilitate it.
- Java programmers have *finally* joined the 21st century and will *occasionally* use non-blocking IO.
- Concurrency is an even bigger part of the language, and a lot of the features from concurrent-first languages have been brought over.

What Changed?

- Since Java 8 and Java 11, there has been a much higher emphasis on functional programming concepts and updated syntax to facilitate it.
- Java programmers have *finally* joined the 21st century and will *occasionally* use non-blocking IO.
- Concurrency is an even bigger part of the language, and a lot of the features from concurrent-first languages have been brought over.

Java 21 New Features.

Java 21 New Features.

Virtual Threads.

Java 21 New Features.

Virtual Threads.

- Virtual Threads are what should have been in Java twenty years ago.

Java 21 New Features.

Virtual Threads.

- Virtual Threads are what should have been in Java twenty years ago.
- Roughly analogous to goroutines in Go.

Java 21 New Features.

Virtual Threads.

- Virtual Threads are what should have been in Java twenty years ago.
- Roughly analogous to goroutines in Go.
- Allow you to have blocking code inside the thread without it breaking the pool.
  - The JVM will park the thread upon seeing a blocking call. . . .
- Extremely lightweight, hundreds of thousands can easily be spun up guilt-free.

Java 21 New Features.

Virtual Threads.

- Virtual Threads are what should have been in Java twenty years ago.
- Roughly analogous to goroutines in Go.
- Allow you to have blocking code inside the thread without it breaking the pool.
  - The JVM will park the thread upon seeing a blocking call. . . .
- Extremely lightweight, hundreds of thousands can easily be spun up guilt-free.
- Implements the same interfaces as regular threads and thus are drop-in replacement.

Java 21 New Features.

Virtual Threads.

- Virtual Threads are what should have been in Java twenty years ago.
- Roughly analogous to goroutines in Go.
- Allow you to have blocking code inside the thread without it breaking the pool.
  - The JVM will park the thread upon seeing a blocking call. . . .
- Extremely lightweight, hundreds of thousands can easily be spun up guilt-free.
- Implements the same interfaces as regular threads and thus are drop-in replacement.

Java 21 New Features.

Java 21 New Features.

Records

Java 21 New Features.

Records

- Much simpler than a class.

Java 21 New Features.

Records

- Much simpler than a class.
- Doesn't require its own dedicated file.

Java 21 New Features.

Records

- Much simpler than a class.
- Doesn't require its own dedicated file.
- Can be pattern-matched.

Java 21 New Features.

Java 21 New Features.

Pattern Matching

Java 21 New Features.

Pattern Matching

- FINALLY! FINALLY!

Java 21 New Features.

Pattern Matching

- FINALLY! FINALLY!
- Can be done inside `if` statements and `switch` cases.