Machine Learning in Java

Adam Pocock
Principal Member of Technical Staff
Oracle Labs, Machine Learning Research Group
4th November 2022

What is Java? (I)

- A compiled, statically typed, memory safe, object oriented language with functional programming features
  - It is natively concurrent & multi-threaded, and has a large set of built-in libraries
  - Recent releases have added support for pattern matching & algebraic data types among many other features
- Java code is compiled to bytecode which runs on the Java Virtual Machine (JVM)
- The JVM provides: JIT compilers that convert bytecode into efficient specialised machine code, garbage collection algorithms, and a wealth of observability and monitoring features
- Compiled Java bytecode runs unaltered on the JVM across many platforms (e.g. x86, ARM, POWER, RISC-V) and OSes (e.g. Windows, Linux, macOS)
- The Java ecosystem contains a library for essentially every task
- And there are many other languages which target the JVM as a runtime

What is Java? (II)

- Java 1.0 was released in 1995, moved to a 6 month release cycle in 2017 after Java 9
- The latest version is Java 19 released in September 2022

- OpenJDK is the reference implementation of Java, it's open source and collaboratively developed
  - It's available under the GPL, and hosted on GitHub – https://github.com/openjdk/jdk
  - Development led by Oracle, with contributions from many other companies and individuals

- Java is widely used across industry to develop large applications, along with many other uses
  - 98 of the US Fortune 100 companies are hiring Java developers
  - Companies built on Java include: Twitter, Netflix, Amazon, Oracle, …
- There are millions of Java developers, and it is consistently one of the most popular languages

What is Machine Learning?

What is Machine Learning?

- Imagine we have an unknown function: `(float[] a) -> float[] b`
- Supervised Machine Learning algorithms take many example pairs (a,b) and create an approximate function which when given a produces b
- Unsupervised ML algorithms take in many examples of a and create an approximation to infer b (e.g. clustering datapoints)
- As simple as an if statement, or as complicated as 100GB of floats
  - Both of them are “machine learning” if the parameters are set automatically based on the data
- This input data is our “training data”, finding the approximation is called “training a model”, and the approximation itself is the “model”

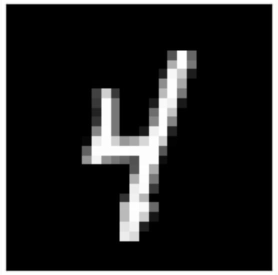
Kinds of Machine Learning

- Supervised learning
  - Learning that these inputs imply a specific output
  - Given a set of (input, output) pairs, learn the mapping function from inputs to outputs
  - E.g., this collection of pixels is a cat, or this piece of text has positive sentiment
- Unsupervised learning
  - Learn patterns in the input even though no explicit feedback is supplied
  - These examples can be clustered together, or this example is different from the others
  - E.g., identifying topics in a set of documents, detecting anomalies in sensor readings
- Reinforcement learning
  - Learn optimal action sequences from a series of rewards
  - E.g., online advertising, playing Chess or Go

A simple Machine Learning model

- Imagine a binary classification problem e.g., classifying if a digit is a 4 or a 9 from the pixel values
- One of the simplest models for this task is a logistic regression

$$p(y) = \sigma(\sum_i x_i * w_i + b)$$



- We have a single learned weight per input pixel, and multiply those weights with the input pixel values, add a learned bias, then pass it through a sigmoid
- If the probability is greater than 0.5 we predict it's a 9, less than or equal we predict it's a 4.
- The weights are randomly initialized, and then updated based on their gradient
  - There are many ways to solve a logistic regression, this is stochastic gradient descent
- Most ML systems have many more trainable parameters per input value, in some cases stretching into the millions

Machine Learning systems are compute intensive (I)

- If we have millions of parameters how do we fit them all?
- Models are compute intensive to train, in some cases requiring 1000s of GPUs for weeks or months
  - This training procedure need only happen once, producing a model which can be reused
- They can also be compute intensive to deploy, models like GPT-3 require at least 5 $10,000+ GPUs and still take several seconds to produce a single output
  - They can run on CPUs but take *much* longer, and still require large core counts with 300GB+ RAM
- When we say "compute-intensive" what is actually being computed?
  - For deep learning models this is many, many matrix multiplications or convolution operations (which can be implemented using matrix multiplication)
  - For other ML models this can be many vector comparison operations, or probing tree structures

Machine Learning systems are compute intensive (II)

- The compute demands of ML systems are now driving the creation of a variety of different hardware
- Modern hardware meets this large compute need in two different ways:
  - Special purpose matrix hardware like Google's TPUs or Nvidia's TensorCores
  - SIMD units which perform the same operation on a vector of values at the same time
- Most modern CPUs have wide SIMD units, allowing them to multiply 16 or 32 numbers together with a single instruction, in roughly the same time the standard multiply instruction on two numbers
- A GPU can be seen as a set of SIMD units where each unit operates on 32 elements at once
- Programming this special hardware can be tricky, it's much lower level than we'd prefer to be
  - We (usually) can't directly program it from high level languages like Python and Java

ML models don’t behave like other code (I)

- The algorithms we use to train ML models are very robust
  - I've implemented many models slightly incorrectly, yet they still learned to solve the training task
  - Just with lower accuracy than the version without bugs
  - That makes it hard to check your training implementation for correctness.
- A machine learning model above a certain complexity is not explainable, and it's difficult to understand what the output will be without executing it
  - They are not specified in the way other programs can be, nor can they be analyzed with correctness proving tools
  - Formally modelling how these algorithms behave on the real numbers is already tricky, with finite precision floating point it’s even harder
- This lack of specification means it's hard to know if a model is applicable to a specific problem without trying it
  - A speech recognition model doesn't know what language it recognizes, that's stored externally.

ML models don’t behave like other code (II)

- ML models usually fail silently
  - If you feed a Spanish speech recognition model English speech, it will produce garbled Spanish text, not an error message saying "I don't understand English"
  - All the model sees are floats, it doesn't know where they came from
- As failures are silent the incoming data can drift away from the trained model
  - A model trained on data from 2019 might not work on 2020 data due to the pandemic
  - The model starts to make more mistakes, but doesn’t flag any of these as incorrect data
- Together these properties mean ML models should be carefully integrated into applications
  - They need careful monitoring and testing, both before deployment and in production
  - Tracking the properties of each model is crucial, you need to make sure that it’s applicable to the data or problem you are trying to solve

Machine Learning software

The ML software ecosystem (I)

- Lots of Machine Learning happens in Python these days
- There are many Python libraries for different tasks, and lots of interesting research on ML models
- However if you analyze the execution time of these programs, very little of it is spent in Python code
- It's all in C/FORTRAN (for CPUs) or CUDA (for GPUs)
  - This is where the convolution and matrix multiply operations that take all the compute time are implemented
  - Native code is the only way to access SIMD or special purpose hardware like TPUs
- Python is merely a wrapper around the native libraries which do all the work
  - And the dynamic nature of Python makes it very easy to get research ideas up and running
  - But the same dynamic nature makes it harder to write large applications
- So when deploying ML in large applications we're more likely to choose a language suitable for building those systems, such as Java

The ML software ecosystem (II)

- There have been ML libraries in Java for almost as long as there has been Java
  - WEKA was one of the first open source ML libraries with development starting in 1997
- Over time academic ML research has moved from MATLAB & Java towards Python, and while much ML research now happens in industry that research continues to be implemented in Python
- Many large applications, particularly in enterprises, are written in Java
  - Lots of large companies (including Oracle) run ML inference workloads on the JVM

- There are two parts to the Java ML ecosystem
  - Single node (deploying models, or training smaller models)
  - Big data ecosystem (scale out data processing on hundreds or thousands of machines)
- All the big data systems of the past 10 years have been built on the JVM (Hadoop, Flink, Cassandra, Kafka, Spark)

Integrating ML into large applications

- Java has traditionally been the language of large complex applications
- Compile time type safety, object oriented design and a high performance VM mean it is the platform of choice for large systems
- You might ask why if ML is a function `float[] -> float[]` do we care about type safety, isn't it all just floats?
- However humans and the programs we write operate on more complex data structures
- We don’t care about float arrays, we want to make predictions for relevant objects:
  - Images, sentences, database tables, etc
- And produce useful outputs:
  - Object detection bounding boxes, annotations for people, places and organizations in text, etc
- These are the inputs and outputs that we care about, and we'd like to use type systems to enforce that the ML is operating on the right data and producing the right outputs
  - Which makes models simpler to integrate into applications

Single node ecosystem

- Java has libraries which fill similar spots in the ML ecosystem as Python's scikit-learn
  - For more traditional ML there are: SMILE, Tribuo, WEKA and several others
- For deep learning there are Java bindings to the same native libraries used in the Python ecosystem
  - TensorFlow, pytorch, MXNet
  - Along with Java specific libraries like DJL and DL4J
- There are also deployment/inference libraries like ONNX Runtime, which can execute ML models, but not create them
- I work on several of these libraries (Tribuo, TensorFlow-Java, ONNX Runtime)
  - But even the ones I don't work on are still good

Tribuo

- Our group in Oracle Labs developed Tribuo to provide ML functionality in Java
  - Open sourced in 2020, GitHub: https://github.com/oracle/tribuo, website: https://tribuo.org
- Tribuo aims to fix some of the problems we found building ML into large applications
  - All the models are self-describing, they record the training metadata, model architecture and parameters inside the model so they are always available in production
  - We wrap the float array interface in something higher level which accepts Examples and produces Predictions
  - Examples have named feature dimensions (to catch mismatches at deployment time) and Predictions have named outputs (to make it easier to get information out of the prediction)
  - Tribuo manages the mappings between floats and names, so it loudly fails if you send image pixels to a sales prediction model, because none of the names will line up
  - Models are typed by their output, so you can't incorrectly use a classification model for clustering
- We have support for deploying models trained in external systems (e.g. TensorFlow, ONNX)

Apache Spark

- Apache Spark is one of the leading frameworks for processing large amounts of data
- Originally focused on batch processing, as a successor to Hadoop's MapReduce
- It now also provides analytics and streaming data operations over large clusters of computers
- Spark integrates SQL like data processing with ML model applications
  - Wrangling data in SQL is a very frequent step before applying ML to tabular data
- SparkML has been developed in conjunction with Spark for many years, and supports a wide variety of ML algorithms (though not many deep learning ones)
- In addition to SparkML models, other models can be deployed and included as part of a data processing pipeline
- Spark also provides pyspark, a python interface to the Spark cluster
  - This pairs each JVM in the Spark cluster with a Python VM to allow models and libraries from Python to operate over the large datasets in Spark

Other big data systems

- Even if the system doesn’t present a Java interface, many big data systems are still built on the JVM
- H2O is distributed ML framework, though it has a proprietary product on top
  - It's written in Java, but the user facing interface is Python or R
  - Integrating new algorithms into it requires working with the Java internals
- ElasticSearch/OpenSearch is a clustered search engine written in Java on top of Apache Lucene, even if most users interact with it via a REST API in the language of their choice
  - Plugging in ML functionality means integrating into the Java internals
  - OpenSearch recently added ML model support using Tribuo & ONNX Runtime
  - They already support using tree ensembles to improve search ranking, using XGBoost4J

Accelerating Machine Learning in Java

Working with native code

- As mentioned earlier, most ML computation happens in native C or CUDA code
  - Accessing this code from other languages is usually done through a "Foreign Function Interface"
- In Java this used to require writing C glue code that translated Java methods and data into C types and called C functions, using the Java Native Interface (JNI)
- Working with native memory can be problematic, as mistakes can crash the whole runtime, so writing JNI code is tricky
- Recent versions of Java are previewing a new FFI, which automatically generates Java bindings for native libraries, and provides a simple and safe way to work with native memory allocations
- As it's still in preview, this hasn't been adopted by many libraries, but it promises to make accessing fast native libraries much simpler, which will benefit the ML ecosystem
  - And as a maintainer of a bunch of JNI code in ML libraries it'll make my life much simpler
  - Which will hopefully result in the libraries getting new features faster

Speeding up computations with SIMD operations (I)

- Using SIMD instructions available in modern processors usually requires writing C or assembly code
- These instructions use wide (up to 512-bit) registers containing multiple values (e.g., 32 floats)
- Writing the code in C directly uses the instruction names
  - E.g., the `_mm512_fmadd_ps(x,y,z)` function inserts the `vfmaddps` instruction operating on the arguments (x,y,z) after loading them into vector registers
- It is tricky to port SIMD code between architectures, each new SIMD instruction set needs new code
  - x86_64 has at least 3 (AVX, AVX2, AVX-512)
  - ARM has 2 (NEON, SVE)
  - RISC-V, POWER etc all have their own SIMD instruction set

Speeding up computations with SIMD operations (II)

- Using this code is crucial for getting the highest performance numerical code, and in ML we want all the floating point operations we can get
- Users need to opt into SIMD code as it has different semantics to regular floating point code
  - Fusing multiplies and adds together gives differently rounded results
  - Floating point operations are not associative, so performing additions in different orders gives different answers
- Thus the compiler tends to be cautious when automatically converting code to SIMD, which means users need to write it directly
  - And writing lots of C with assembly instructions inside it is complicated

Java Vector API

- For the past few years the Java platform has been building a cross-platform API which allows users to use SIMD instructions directly from Java code
- Users say they want to add two vectors together, and the JVM picks the register size and instruction based on the CPU it is running on
- We can write SIMD code at a higher level and have it run on x86, ARM etc without any further coding
- The API initially appeared as an incubating feature in Java 16 in 2021, and has been improved in successive releases
  - It's not finalized yet, but the performance improvement is impressive for the right workloads
- I've been working with the early prototypes available in OpenJDK Project Panama since 2018
  - We ported several ML models to test out the functionality

A simple Vector API example

- As mentioned earlier the dot product is a simple operation which underlies a lot of ML computations

$$x.w = \sum_i x_i * w_i$$

- We multiply each element of the inputs together, and sum the result
- The straightforward implementation of it using scalar operations is:

```
private static float scalarDot(float[] first, float[] second) {
    float accumulator = 0.0f;
    for (int k = 0; k < first.length; k++) {
        accumulator += first[k] * second[k];
    }
    return accumulator;
}
```

Implementing a dot product with C intrinsics

- We can implement the same operation using AVX-512 intrinsics in C
  - This will only run on CPUs which support AVX-512 (AMD Ryzen 7000s, recent Intel Xeons)

```c
float dotAvx512(float* first, float* second, size_t length) {
    // accumulate dot product using SIMD
    __m512 buffer;
    int i = 0;
    for (; i < (length & ~15); i += 16) {
        __m512 firstVec = _mm512_loadu_ps(first + i);
        __m512 secondVec = _mm512_loadu_ps(second + i);
        buffer = _mm512_fmadd_ps(firstVec, secondVec, buffer);
    }
    // reduce buffer to scalar value
    float output = _mm512_reduce_add_ps(buffer);
    // accumulate tail of vector
    for (; i < length; i++) {
        output += first[i] * second[i];
    }
    return output;
}
```

Implementing a dot product with the Vector API

- Compared to the C version, we operate on higher level constructs
  - The `VectorSpecies` object lets the JVM pick the vector size and instructions

```
static final VectorSpecies<Float> FLT_SPECIES = FloatVector.SPECIES_PREFERRED;
private static float simdDot(float[] first, float[] second) {
    // accumulate dot product using SIMD
    int i = 0;
    FloatVector buffer = FloatVector.zero(FLT_SPECIES);
    for (; i < FLT_SPECIES.loopBound(first.length); i += FLT_SPECIES.length()) {
        FloatVector firstVec = FloatVector.fromArray(FLT_SPECIES, first, i);
        FloatVector secondVec = FloatVector.fromArray(FLT_SPECIES, second, i);
        buffer = firstVec.fma(secondVec, buffer);
    }
    // reduce buffer to scalar value
    float output = buffer.reduceLanes(VectorOperators.ADD);
    // accumulate tail of vector
    for (; i < first.length; i++) {
        output += first[i] * second[i];
    }
    return output;
}
```

Comparing the implementations

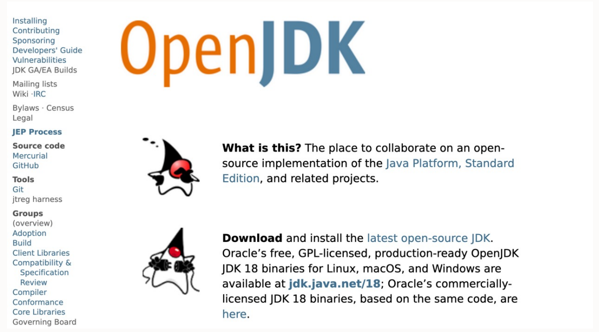
- Unlike C, the Java implementation will use available SIMD instructions on x86 chips without AVX-512 and even the SIMD instructions on ARM without modification
- So it requires little effort to get accelerated numeric code across many different platforms

- Fast native implementations of numerical operations tend to be specialised for each possible hardware combination
  - If we know the array is 48 elements, then we may want to process it with three 16 element (i.e. 256-bit) vectors even if 32 element (i.e. 512-bit) vectors are available
  - As using 32 element vectors would mean we need to cope with a 16 element tail anyway
- Consequently it is possible in the Vector API to manually specify the vector size, at the cost of a slowdown if the hardware doesn't support that size
  - But most of the time the JVM can automatically pick a good enough option

Conclusions

- We discussed what kind of computation Machine Learning is, and how it fits into larger applications
  - And why that integration affects how the ML model is built
- We discussed a few of the options for building and deploying ML models in Java
  - And why using a more typed system improves integration with large codebases
- Finally we looked at how the Java platform is evolving to better support heavy numerical workloads
  - Improving the foreign function interface makes it easier to access efficient native libraries for matrix multiplication and to access accelerator hardware like GPUs
  - Adding SIMD support to the language means some of those efficient numerical algorithms can be implemented directly in Java, making them portable and easier to maintain
- Together these improvements allow more of the ML system to be written in a higher level language (from C/Fortran to Java) making it easier to deploy ML workloads across platforms

Learn more about Java



Installing
Contributing
Sponsoring
Developers' Guide
Vulnerabilities
JDK GA/EA Builds

Mailing lists
Wiki ·IRC

Bylaws · Census
Legal

JEP Process

Source code
Mercurial
GitHub

Tools
Git
jtreg harness

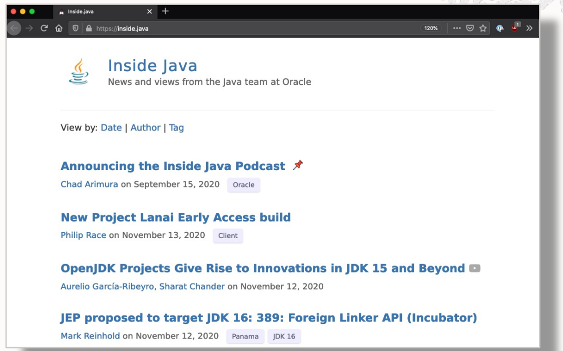
Groups
(overview)
Adoption
Build
Client Libraries
Compatibility & Specification Review
Compiler
Conformance
Core Libraries
Governing Board

OpenJDK

What is this? The place to collaborate on an open-source implementation of the Java Platform, Standard Edition, and related projects.

Download and install the latest open-source JDK. Oracle's free, GPL-licensed, production-ready OpenJDK JDK 18 binaries for Linux, macOS, and Windows are available at **jdk.java.net/18**; Oracle's commercially-licensed JDK 18 binaries, based on the same code, are here.

Openjdk.org



Inside.java

https://inside.java

Inside Java
News and views from the Java team at Oracle

View by: Date | Author | Tag

Announcing the Inside Java Podcast
Chad Arimura on September 15, 2020 Oracle

New Project Lanai Early Access build
Philip Race on November 13, 2020 Client

OpenJDK Projects Give Rise to Innovations in JDK 15 and Beyond
Aurelio García-Ribeyro, Sharat Chander on November 12, 2020

JEP proposed to target JDK 16: 389: Foreign Linker API (Incubator)
Mark Reinhold on November 12, 2020 Panama JDK 16

Inside.java

Java Learn Download Community Contribute News Future

The Arrival of Java 17

Used by over 10 million developers and running on 56 billion devices globally, the Java Platform truly moves the world forward, and now even faster with Java 17. Download or try out Java here!

Download Java 17 Learn Java

Currently Java 17 · Previous releases

The Destination for Java Developers

Hundreds of tutorials, news and videos from the experts, all right here.

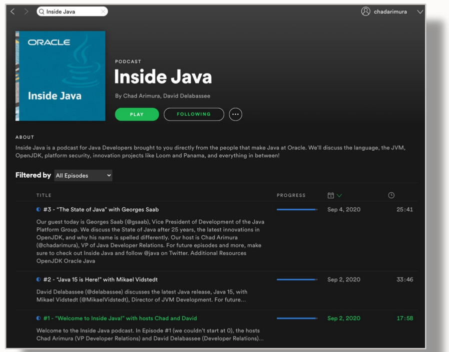
Get Started
- Getting Started with Java
- Java Language Basics
- Downloading Java

Go Deeper
- Lambda Expressions
- The Stream API
- The Collections Framework

Resources
- Java News
- Official Java Podcast and More
- Upcoming Events

Dev.java

youtube.com/java



Inside Java

chadarimura

PODCAST
Inside Java
By Chad Arimura, David Delabassee

PLAY FOLLOWING

ABOUT
Inside Java is a podcast for Java Developers brought to you directly from the people that make Java at Oracle. We'll discuss the language, the JVM, OpenJDK, platform security, innovation projects like Loom and Panama, and everything in between!

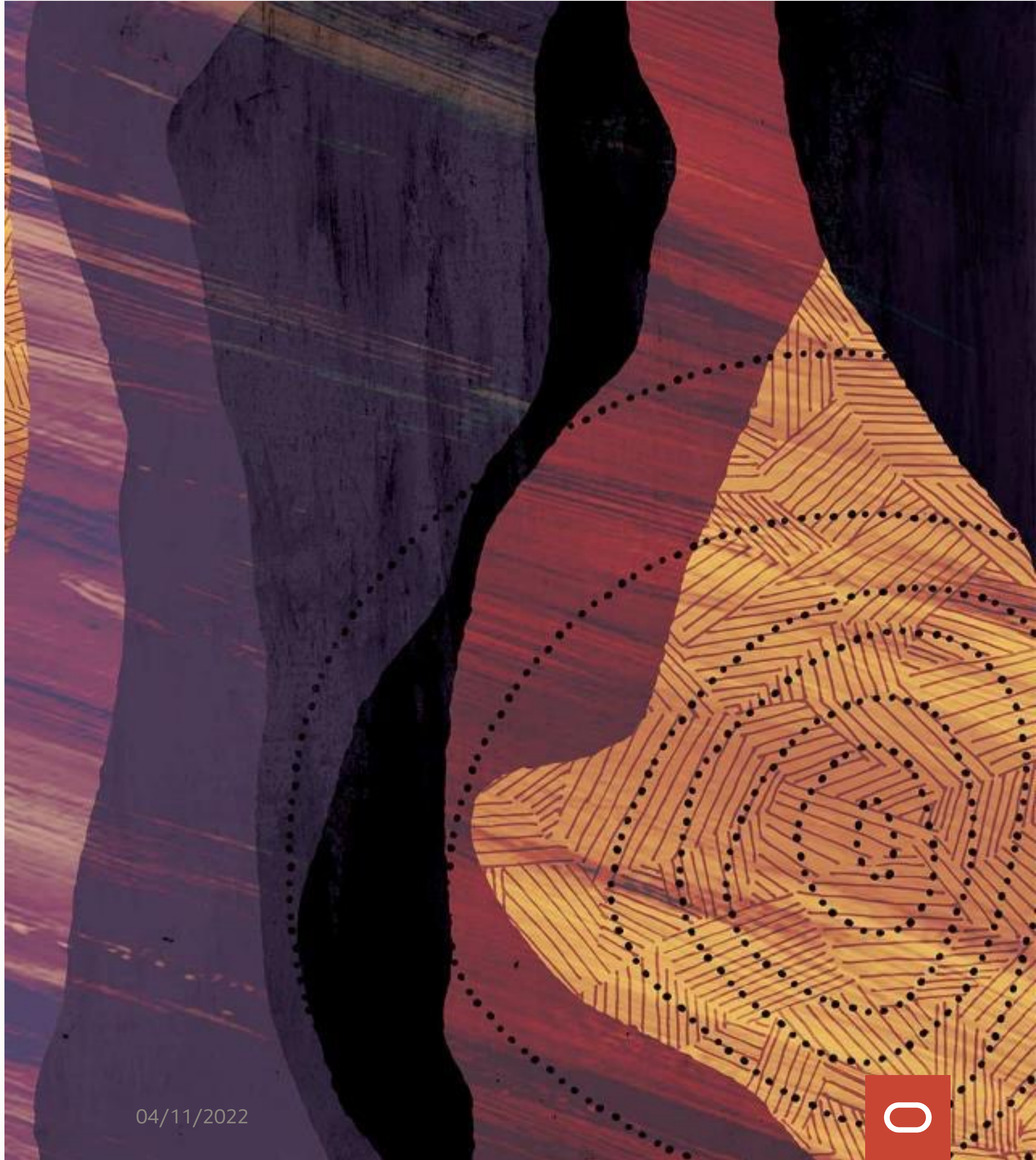
Filtered by All Episodes

TITLE	PROGRESS		
#3 - "The State of Java" with Georges Saab Our guest today is Georges Saab (@gsaab), Vice President of Development of the Java Platform Group. We discuss the State of Java after 25 years, the latest innovations in OpenJDK, and why his name is spelled differently. Our host is Chad Arimura (@chadarimura), VP of Java Developer Relations. For future episodes and more, make sure to check out Inside Java and follow @java on Twitter. Additional Resources OpenJDK Oracle Java		Sep 4, 2020	25:41
#2 - "Java 15 is Here!" with Mikael Vidstedt David Delabassee (@delabassee) discusses the latest Java release, Java 15, with Mikael Vidstedt (@MikaelVidstedt), Director of JVM Development. For future...		Sep 2, 2020	33:46
#1 - "Welcome to Inside Java!" with hosts Chad and David Welcome to the Inside Java podcast. In Episode #1 (we couldn't start at 0), the hosts Chad Arimura (VP Developer Relations) and David Delabassee (Developer Relations)...		Sep 2, 2020	17:58

Inside Java Podcast

Thank you

Any Questions?



ORACLE

Our mission is to help people see data in new ways, discover insights, unlock endless possibilities.