Java still rocks the finance industry. Here’s why Java 16 makes it even better.

High-speed trades? Quantitative analysis? The investment pays off.

by Alexa Morales

With the advent of retail trading and a bull market running for more than a decade now, the stock market has never been more popular. In fact, according to Bloomberg Intelligence, retail investors now dominate US equity markets, nearly accounting for as much volume as mutual funds and hedge funds combined. There’s also interest in so-called meme stocks, thanks in part to the WallStreetBets forum on Reddit and the recent attempt at a short squeeze on the struggling retailer GameStop.

The career prospects for quants and financial engineers are soaring. Prestigious Master of Financial Engineering degree programs churn out new generations of technologists wielding the programming tools to make their trading algorithms shine. Let’s not forget the pandemic-driven market crash of early 2020, either. That proved the resilience of the systems, says David Mercer, CEO of London’s LMAX Group: “Just in 2020 our bank volumes doubled, and most of that comes from algo desks—so now about 85% of bank trading volumes come through algo desks.”

That brings up the state of the art in financial engineering—and the reasons why Java is still critical to how stock markets run. But first, a little history.

Java's long history in finance

In 1969 Instinet became the first electronic communication network (ECN) for trading financial products beyond the stock market floor. In the 1970s, computerized order flow dawned, while the 1980s saw the
introduction of Michael Bloomberg's eponymous terminal. In the early 1990s, the now-dominant Financial Information eXchange (FIX) protocol began as a messaging experiment between Salomon Brothers and Fidelity.

By the late 1990s, programmer Joshua Levine, who had been plying his software to different Wall Street companies after dropping out of college in the 1980s, created a small-order matching system called Island ECN. Island, written in FoxPro for MS-DOS version 2.6, had a nifty trick for enhancing speed: Before creating a new order, check to see if there’s a record from a canceled order remaining in the cache. Then, resurrect and reuse that record to avoid a bunch of work. Island was quickly adopted by the Nasdaq.

In 2002, Instinet bought Island and a year later, it ported the code to Java. On his blog, Levine explains: "While the Java version is some *very* nice code and ran several orders of magnitude faster—this [original] code is particularly interesting precisely because it ran on such a slow platform and is so easy to read, highlighting the idea that it is usually the architecture and algorithms that matter more than raw platform speed."


Java was not only much faster: By the early 2000s, Java was revolutionizing programming, thanks to its ease of use, security, portability, dynamic class loading, runtime thread creation, and more.

The high-frequency trading (HFT) race was roaring, while Java-based tools such as thinkorswim (now owned by TD Ameritrade) were making algorithmic trading accessible to everyone.

In 2004, Goldman Sachs began building a Java collections framework called Caramel, which was open sourced in 2012 and later renamed Eclipse Collections. By 2011, an Oracle technical brief estimated that more than 80% of electronic trading applications, and nearly all FIX engines, were written in Java.

The language's popularity created a virtuous cycle for financial companies: “Developers of Core Java are widely available in London and other financial centers, allowing HFT firms to pick the best staff for their needs,” says Peter Lawrey, a Java Champion and CEO of London-based Chronicle Software.

New kinds of exchanges, enabled by Java, were also popping up. Written in Java in 2010, the Chicago Climate Exchange was designed for trading carbon dioxide offsets. Around the same time, LMAX Disruptor, which is still popular today, pioneered new ways of speeding foreign exchange (forex) trades using such techniques as “mechanical sympathy”—an understanding of how CPUs use their memory in concurrently threaded processes.
Another revolution was brewing: the open source software movement. As many Java Magazine readers know, in 2007 Sun Microsystems released the OpenJDK under the GNU General Public License. The ensuing years saw a blossoming of open source stock trading projects.

Today on GitHub, you can find such powerful open source Java-based systems as Exchange-Core, Ta4j, Coin Trader, and Chronicle Queue, which, according to Lawrey, is “widely used by tier-one banks in the world for low latency persistence (around a microsecond) and messaging between microservices for trading systems.”

Other Java quant favorites include:

- **Strata**, an analytics and market risk library
- **JQuantLib**, a quantitative finance framework
- The **finmath library** of mathematical finance algorithms
- The clearly named **QuantComponents**
- **DRIP**, whose libraries cover fixed income, asset allocation, transaction cost analysis, and XVA (risk) metrics

### Java’s evolution fits financial markets

Almost as soon as Java came on the trading scene, both the language and the runtime environment began changing to meet the finance industry’s unique demands. The first biggie was real-time performance.

“**Real-time**, strictly, refers to a guaranteed worst-case response. Most applications specify performance requirements in general statistical terms such as ‘90% of requests should be completed in under two seconds,’” says Simon Roberts, a former Java instructor at Sun. “A real-time requirement might not mandate ‘fast’ in the usual sense (though it often does). What matters is that absolutely no request takes longer to complete than some specified maximum.”

But Java’s original garbage collector wasn’t optimized for real-time operations, and it was prone to halting all processing while it cleaned up unused memory. Thus, in 1998, the first JSR ever issued was for a real-time version of the language, libraries, and JVM, Roberts explains. Implementations of **JSR 1** “are alive and well,” he says. “However, most applications are served well enough by the enormously improved garbage collectors such as G1GC from about Java 8 and more recently ZGC, Shenandoah GC, and others.”

Indeed, the following two new Java 16 features released in March 2021 make Java even faster via memory management:

- **JEP 376**: ZGC with concurrent thread-stack processing moves ZGC thread-stack processing from safepoints to a
In addition, JShell, introduced in 2014, makes it easy to prototype Java code via the command line, which is useful for deadline-driven quants.

A Java 9 innovation welcomed by the speed-obsessed finance industry was JEP 295, ahead-of-time (AOT) compilation, which helps the JIT compiler warm up completely and avoid the performance penalty of invoking infrequently used methods. (Full AOT support came later via GraalVM Native Image.)

Enter GraalVM, Slang, and Truffle

The newest behind-the-scenes technology making AOT (and a lot of other things) possible is GraalVM, which can speed up languages including Java, Python, Ruby, JavaScript, R, C, and C++. But what if your team's language isn't on this list—perhaps because you wrote it yourself? That was the problem facing the Goldman Sachs Securities Database (SecDB) Architecture team, which maintains the company's pricing platform.

The SecDB architects support an in-house programming language called Slang, which is critical to pricing, risk, and trade booking. "I'm going to set the scene of why we've got our own in-house programming language—who's seen Wayne's World?" asks Jackie Haynes, a managing director and tech fellow at Goldman Sachs as well as a JCP Executive Committee member, in a talk at Oracle Code One.

"So it's 1992, and at Goldman Sachs we're working in foreign exchange. We want to code in a dynamic language. We've got clients on the phone; we want to be able to prototype pricing models pretty quickly. And we've got quite a small team so I'm going to be sharing code across different asset classes. In 1992, Python doesn't exist yet," Haynes says. Instead, they created Slang (short for “Securities Language”), an interpreted, single-threaded language. "Syntactically it's the love child of C and Lisp, but with spaces and variable names, because we wanted it readable."

Fast-forward 25 years, and Haynes' team is now trying to speed up those 150 million lines of business-critical Slang code. In her talk, she describes their experimental use of GraalVM and Truffle to run Slang as a speedier JVM language—that Java developers could also maintain.

The Truffle language implementation framework is an open source library for building tools and programming languages
implementations as interpreters for self-modifying abstract syntax trees.

“Slang is heavily used, and as a dynamic language it is not as fast as other languages. So obviously any improvement in performance would be another awesome big win for us,” Haynes says.

During the proof-of-concept, Goldman Sachs discovered that GraalVM was able to provide a base for Slang, and they could switch between Slang and C++: “If that is all running on GraalVM, then you do get an integrated environment where you can profile and debug across those boundaries. That’s just a huge efficiency that was very cool,” Haynes adds.

Further, in that same talk, Haynes’ colleague Zachary Fernandez notes that GraalVM’s LLVM runtime, called Sulong, “kind of blew my mind with a little bit of magic. Writing a class, compiling it to LLVM bitcode, handing that pointer off to our existing interpreter, and then that interpreter being able to call member functions, which would end in Sulong-managed code—that was all very cool.”

**Citadel seeks Java quants**

Ultimately, the finance industry’s programming world is a lot more polyglot than when Java was the hip new language—and that’s a good thing. Thanks to simpler syntax and a plethora of data science libraries, languages such as Python and R, not to mention Haskell and Julia, have exploded in finance.

That said, jobs at market-makers and hedge funds typically still require strong Java and C/C++ skills, not only to maintain existing code but also to circumvent the limitations (such as the Python Global Interpreter Lock) of single-threaded languages, to create performant systems, and to push forward into burgeoning financial realms such as deep learning and blockchain.

Asked whether he’ll switch to other languages and libraries for blockchain or machine learning, Chronicle Software’s Lawrey remains true to Java: “We have developed our high-throughput blockchain in Java. In our opinion, this makes it easier to extend.”

Java boasts a vast ecosystem of runtime options, profilers, debuggers, development environments, and dependency management tools. There’s also a large talent pool, he says: “While some newer languages like Go, C#, and Kotlin have significant interest, Java developers are more widely available, making it a more reliable development environment to use long-term.”

What if you’re looking for a cushy Java developer job in the finance industry? Using Citadel Securities as an example, the
company’s openings ask for expertise in Java SE and Jakarta EE; software patterns and enterprise architecture; and experience with Spring, Hibernate, the FIX protocol, Maven, Git, Jenkins, Nexus, relational databases, and big data.

For any high-frequency trading application, you want to show that you can write memory-efficient code, advises Lawrey: “Value-based classes improve memory footprint and thus the efficiency of CPU caches. While most modern computers have gigabytes or even terabytes of memory, they still have only 256 KB of level 2 cache local to each core, so if you want each core to run as fast as possible, you want to avoid going off your L2 cache—making memory still a precious resource.”

And you’ll need to answer questions about collections, multithreading, and search and retrieval algorithms, such as the relative speed of using binary trees, ArrayList, and vectors. You’ll also want to understand standard finance algorithmics for momentum, arbitrage, mean reversion, and statistical arbitrage.

What's next for Java in finance?

Considering the very recent history of modern popular finance, commission-free trading smartphone apps such as Robinhood aren’t that surprising. After all, the first index portfolio was created in 1971. Vanguard founder John Bogle introduced his S&P 500 mutual fund in 1975. Quantitative investing took off in 1980. The first exchange traded fund was in 1992. The world's stock markets may date back to the 1400s, but computerized finance is brand new.

What looms on the horizon, then? Seven-day trading; cryptocurrency expansion, as it becomes the new forex; green, energy-efficient ways to mine cryptocurrencies; blockchain for secure, frictionless transactions; and deep learning to anticipate the human desires that make stocks go up and down; and maybe—a bit of hope—new exchanges that incentivize planet-friendly practices. Whatever comes next, Java will stay deep in the world of finance. You can take that to the bank.

Dig deeper

- Understanding the JDK's new superfast garbage collectors
- From the vector API to records to elastic metaspace, there's a lot packed into Java 16
- The 25 greatest Java apps ever written

Alexa Morales

Alexa Weber Morales is director of developer content at Oracle. Morales is the
former editor in chief of *Software Development* magazine and has more than 15 years of experience as a technology content strategist and journalist.