

## Can Oracle be trusted with Java?

By Jeff Gould, CEO & Director of Research, Peerstone Research

Published in Open Enterprise News, June 13, 2009

### Why is Java worth so much?

“Java is one of the computer industry’s best-known brands and most widely deployed technologies, and it is the most important software Oracle has ever acquired.”

Thus spake Larry Ellison in April as he snatched Sun Microsystems from the waiting jaws of IBM.

These words are worth parsing carefully, for they convey a claim that is really quite remarkable. What Oracle is buying with Java, says Ellison, is more important than any of the dozens of software companies it has purchased in recent years. More important than Java application server market share leader BEA. More important than CRM top dog Siebel. More important even than ERP heavyweight PeopleSoft, whose acquisition sparked a titanic antitrust battle between Oracle and the Department of Justice. Let’s see. If memory serves, Oracle spent a total of \$24.5 billion on those three acquisitions. That’s quite a bit more than Oracle is paying for Sun. In fact, considering that Sun has agreed to sell itself to Oracle for \$5.6 billion (net of cash and debt), it seems Larry is getting his hands on Java for only a fraction of what he thinks it’s actually worth. From this perspective, the software Deal of the Year begins to look like the Steal of the Decade.

### What is Java, really?

So what exactly does Larry Ellison think he is buying with Java? There is no simple answer to this question, and giving even a half-way complete answer would require telling a very long story, running to thousands of words more than I have space for here. So what follows will by design be a very brief history and explanation of Java.

Of course today Java has so deeply penetrated into the world’s computing infrastructure that it is hard to remember a time when it wasn’t the all-pervasive force it is today. Before we look at the early days of Java, it’s useful to recall a few basic numbers about Java’s worldwide reach, which extends all the way from electronic gadgets clutched in the hands of consumers to the sprawling data centers that run the world’s corporations. On the consumer side, Java now runs on some 6 billion devices, [including 2.5 billion cell phones](#). Yes folks, that’s billion with a “b.” By comparison, Microsoft Windows runs on a measly billion or so PCs. In the enterprise, Java is the dominant server-side application platform. Enterprise Resource Planning software (ERP) from SAP and Oracle, online Customer Relationship Management (CRM) from Salesforce.com, giant customer portals for the world’s biggest consumer brands, trading applications on Wall Street, and countless custom business applications in companies large and small all run on Java.

But still the question remains, what is Java exactly? Well, first and foremost, it is a programming language, originally developed by James Gosling at Sun in the early 1990s for use in set-top boxes. Java is said to be a simpler and smarter version of C and C++, at that time the favorite system programming languages of the Unix and Microsoft (though not the IBM) software worlds. But Gosling and Sun were after something more than just a better way to write code for devices that sit on your TV. They realized

## Can Oracle be trusted with Java?

Java had the potential to be something much greater. And they also wanted to change the rules by which the computing game had been played until then. They wanted a programming language that would be hardware and operating system independent. The famous slogan that launched Java was “write once, run anywhere,” or WORA. Java code would be compiled, like most programming languages, but the compiled code would run, not on actual hardware, but on a virtual machine implemented in software. This virtual machine (written, as it happens, not in Java but mostly in C and C++) would in turn be ported to all the major hardware and operating system platforms, thus fulfilling the WORA promise.

Although the Java virtual machine (JVM) introduced an extra layer of complexity that made early Java code run noticeably slower than “native” apps written in C or C++, Sun executives nevertheless touted Java’s platform independence as a key advantage, because they saw it as a way of countering the dominance of Microsoft Windows. Applications written to the Windows programming interfaces (APIs) installed on Intel-based PCs couldn’t easily be ported to competing platforms. Applications written for the JVM, on the other hand, could go anywhere. Thus a Java application could in theory run on a Windows-based PC, a RISC-based Unix server, a Macintosh, a mainframe, or ultimately even on a cell phone. Of course there were some restrictions to this rule, mostly having to do with whether the application’s intended role was that of a back-end server or a user-facing client. The latter would require user interface features, while the former would need to handle things like database access and multiple client sessions.

The next big leap in Java’s history was largely a question of being in the right place at the right time. By the latter half of the 1990s the Internet boom was in full swing. Web sites were rapidly evolving from simple HTML pages to full fledged e-commerce sites where goods and services were bought and sold. These sites needed to store their sales catalogues and transaction records in relational databases. Furthermore, they needed sophisticated code to guarantee the integrity of the sales and reservation transactions they were executing in huge numbers. Smart entrepreneurs quickly spotted the opportunity. Soon, a new form of web-oriented transaction middleware emerged, the confusingly named “application server” (confusing, because the name made this purely software product sound like it was a piece of hardware).

Software startups created application servers using all kinds of programming languages, some of them established, like C++, some of them made up for the occasion, like ColdFusion Markup Language. But a few especially clever startups thought to borrow Sun’s newly invented Java, which at the time Sun made freely available to all comers. Two of the most prominent Java startups were quickly snapped up by bigger companies, NetDynamics by Sun itself and WebLogic by BEA, both in 1998. At the same time, by coincidence, a lone programmer at Sun was working on a lighter weight Java application server which he named Tomcat. Sun’s management, deciding that Tomcat didn’t have much commercial potential, quickly handed it off to the open source Apache Software Foundation. Last but not least, the potential of Java application servers had not gone unnoticed at the dominant vendor of transaction processing platforms, mainframe giant IBM, whose programmers launched WebSphere in that same eventful year of 1998.

### **JCP: the standards body that isn’t**

The extraordinarily rapid ascent of Java – and in particular of the Java application server as the defining piece of web middleware – took everyone by surprise, including Sun. In its early years Java evolved without any real standards other than a few early APIs proposed by Sun, such as JDBC for database access and JSP for Java Server Pages (the Java world’s equivalent of Microsoft’s Active Server Pages). But by the late 1990s Sun’s management realized this situation was untenable. Without some kind of standards framework to control the market and limit excessive competition over new features, the coalition of big league software vendors who had lined up with Sun to back Java against Microsoft might

## Can Oracle be trusted with Java?

splinter. Sun briefly considered trying to make Java a formal standard by submitting it to the International Standards Organization (ISO) and then ECMA, but soon changed its mind. Instead, it decided to create its own standards vehicle, one that would be more amenable to its control. Thus was born in 1998 the Java Community Process, or JCP. In addition to Sun, the JCP numbered among its founding members both BEA and IBM, soon joined by Oracle. Later joiners included SAP, Google, HP, Intel, Red Hat and most of the world's leading cell phone manufacturers, among which Nokia, Sony-Ericsson, Motorola and RIM.

The most important thing to understand about the JCP is that it is not in fact a standards *body* in the usual sense of the term. It is not a non-profit foundation like Apache or Eclipse. It is not a public entity like the IETF, much less a quasi-governmental organization like the ISO. Nor is it a vendor-neutral membership consortium like Oasis or the W3C. In fact, as Java developer and Apache member Stephen Colebourne has pointed out, the JCP is not really an organization at all. As its name suggests, it is merely a "process," one that is largely though not entirely controlled by Sun.

In the strict sense, the Java Community Process has no legal existence at all. It is nothing more than a collection of bilateral contracts between the separate members of the JCP (who may be individuals, corporations, or other standards groups) and Sun. These legal agreements, which will be inherited intact by Oracle when the acquisition is complete, are known as [Java Specification Participation Agreements](#) (or JSPAs). They lay down the process by which Java standards (known as Java Specification Requests, or JSRs) are created as well as the rules governing the all-important exchange of intellectual property (patents, copyrights, trade secrets) required of all JCP members who wish to work on Java standards. Anyone who wants to understand how this process works in detail should study the [highly instructive account](#) by Stephen Colebourne in his blog, which is accompanied by helpful diagrams and links to the original materials.

The day-to-day affairs of the JCP are managed by the Java Program Office, which is staffed by Sun employees and answerable only to Sun management. The official direction of the JCP is entrusted to two Executive Committees, one for the Standard and Enterprise editions and one for the Micro Edition. But here too Sun remains the power behind the scenes. Of the 16 seats on each Committee, one is held permanently by Sun (the only member to enjoy such a privilege), 10 are filled by candidates that Sun nominates and then submits to the JCP membership for ratification, while only the remaining 5 are filled by unrestricted vote of the membership. The JCP is thus entirely Sun's creature, but it is not entirely subject to Sun's every whim. Rather, it is an ingenious mechanism crafted over the years to manage conflict among competitors. By providing a semblance of participatory democracy for the mass of JCP members and just enough real power for the few dozen heavyweight vendors who really count, Sun has been remarkably successful in keeping a very fractious group of industry players on the Java reservation.

It is only fair to acknowledge that Sun's creature has been admirably productive. Once the JCP got off the ground in the late 1990s it began pounding out standards (JSRs) thick and fast. These include the specs for the Java language itself, the virtual machine, and the compilers, together with a vast collection of APIs and libraries that sit above the virtual machine and constitute the various "editions" of the so-called Java platform. These libraries define the rich off-the-shelf functionality that certified Java application servers must provide. Over time three distinct editions have evolved, addressing different target audiences:

- Java Standard Edition (Java SE) is the basic set of libraries that can be used by clients or servers;
- Java Enterprise Edition (JEE) augments SE with additional libraries used by high-end enterprise server applications;

## Can Oracle be trusted with Java?

- Java Micro Edition (JME) is a related set of libraries for small footprint devices like cell phones or PDAs.

(Note: Some people may be surprised by the absence of the well-known J2EE from this list. But the number “2” in Sun’s Java nomenclature was an obscure historical convention that was dropped in 2006. The terms J2EE and J2SE are thus now obsolete.)

Simply using the Java language to write programs (for example, in a college programming class) is not the same as using the Java platform. Using the platform in one of its three editions means deploying the Java Runtime Environment (JRE), which consists of the JVM and one of the library collections just mentioned. Developing programs for the Java platform requires the JDK (Java Development Kit), which adds compilers and other developer tools to the JRE (Wikipedia has an [excellent diagram of all the pieces and layers in the Java stack](#)). Java application server vendors pay license fees to Sun for the intellectual property in the JRE, but they are also free to add their own proprietary features and functionality, and all serious commercial Java vendors do.

Indeed, a commercially viable application server has to offer a lot more than just compliance with the JCP-sponsored standards. Over the years Java vendors like BEA, Oracle, IBM, JBoss and Sun itself have sought competitive differentiation for their products in certain key areas of runtime functionality and performance. Java application servers can be compared to cars and trucks. Thanks to the JCP they all have a similar set of controls that developers can use, the equivalent of brakes, steering wheels, accelerator peddles, dashboard controls, and so forth. Just as vehicles differ in what lies under the hood, what the body looks like, or what kind of covers the seats have, so application servers differ in such features as clustering, load balancing, failover, performance tuning, transactional integrity, security, and manageability. Just as anyone who knows how to drive will be able to operate more or less any kind of road vehicle with wheels and an engine, all Java application servers offer roughly the same set of code-facing features and standards. But under the hood Java app servers can be as different from each other as dump trucks, Ferraris, and VW minivans. The one key thing all Java application servers have in common, however, is their great complexity. The effect of the JCP and its sprawl of specifications, many of which are very complex in their own right and all of which in a given edition must be implemented to secure certification, is to guarantee that no upstart tinkerer can build a simpler Java mousetrap on his or her own and then launch it into the marketplace in defiance of Sun and the JCP.

### How Sun uses the JCP to control Java

A very significant peculiarity of the JCP is that, unlike other standards groups, it doesn’t limit its output to paper specifications, but also produces for each standard an actual implementation (the “reference implementation” or RI) as well as, crucially, a compatibility testing kit. Here is where the Java standards creation process gets a little hinky. Each JSR starts with a Spec Lead, typically a software or system vendor interested in leading the development of some particular Java standard. Historically Sun has served as Spec Lead for the most crucial JSRs, those covering the virtual machine and the major platform editions mentioned previously. But the total number of JSRs now runs into the hundreds, and many other vendors besides Sun have played the role of Spec Lead for one JSR or another. The first job of the Spec Lead is to recruit a number of like-minded participants to form an Expert Group for the JSR. The group then sets to work hammering out the paper specification and building the reference implementation. The members of the Expert Group agree to grant all necessary intellectual property rights (copyrights, patents, trade secrets, etc.) to each other and to the Spec Lead. Finally, when the work is finished, the Spec Lead develops the all-important test kit. The latter, known as a Technology Compatibility Kit (TCK), is a suite of programs that test any proposed implementation of the new Java

## Can Oracle be trusted with Java?

standard (JSR) to establish that it fully and correctly implements the spec with all required interfaces and functionality. If an implementation can't pass the TCK, it doesn't earn the right to claim compatibility with the JSR.

I'll admit that the preceding paragraph is a little on the inside baseball side. But it all makes sense, right? Wait just a minute, though, there are a few surprises lurking in the fine print. The JSPA agreement signed between all JCP members and Sun states that it is the Spec Lead for each JSR who controls the licensing of that JSR's TCK. Only when a given implementation has passed the relevant TCK does it receive the IP rights to that implementation contributed by the members of the Expert Group. The developer of a new candidate implementation must secure a license for the TCK code from the Spec Lead, and this license is not necessarily free. Although only the interested parties know for sure, it is widely believed that major Java application server vendors like IBM, Oracle and Red Hat pay hefty fees to Sun for the TCK licenses they need to certify their software as compliant implementations of the Java standards. (A TCK for certifying compatibility with a platform JSR, e.g. for something like Java Enterprise Edition, is often known as a Java Compatibility Kit, or JCK.)

But stiff licensing fees are not the only encumbrance that can come with a TCK. While the JSPA in theory requires the Spec Lead to offer the TCK license under "reasonable and non-discriminatory" terms, there have been some notable departures from this rule. The exceptions seem to involve cases where Sun itself is the Spec Lead and has a strategic interest in placing roadblocks in front of competitors. One prominent victim of Sun's ability to leverage TCK and JCK licensing rights is Apache's Harmony project. Backed by IBM, Intel and Google among others, Harmony is an attempt to recreate as open source code the entire Java SE platform, including the virtual machine and all of the libraries. When, after unexpectedly rapid progress toward this goal, the Harmony group asked Sun for a JCK license, Sun nominally agreed, but then proceeded to add certain "field of use" restrictions to the license that limited the marketability of any software products based on Harmony.

Many key details of this dispute are hidden behind a veil of non-disclosure agreements, but it appears that Sun has been seeking to prevent the use of Harmony in cell phones, where it would compete with Sun's lucrative Java ME franchise. This interpretation is supported by the interesting fact that Google's Android operating system for cell phones makes use of the Harmony class libraries. One can also speculate that if Sun were to allow the certification of Harmony as a fully compliant implementation of the Java platform, it would ultimately threaten Sun's ability to extract license fees from and exercise strategic leverage over the major enterprise Java players, i.e. IBM, Oracle and Red Hat. (Readers who want to know more about the Sun-Harmony dispute should consult the meticulously detailed account in Stephen Colebourne's blog.)

The legal wiggle-room that Sun accords itself in granting or not granting TCK licenses, or in choosing to impose restrictions on certain licensees, is widely understood in the Java community to be the decisive way in which Sun exercises its strategic control over Java. It is this hidden legal power that Oracle is about to inherit, the power to control not only what software has the right to call itself Java (or Java EE, Java ME, etc), but also which competitors' products will receive the indispensable IP rights without which no commercial software can survive.

### **But doesn't open source Java change everything?**

But wait, you say, what about open source? Doesn't Sun's surprising and dramatic decision to release the source code for Java under the famous GPL license – the very same which governs Linux – mean that all

## Can Oracle be trusted with Java?

these shadowy maneuvers over intellectual property rights and TCK licenses are a thing of the past? The answer is no, not quite... or perhaps even, not by a long shot.

You see, Sun remains the copyright owner of the code in the JRE, the JDK and the libraries in the various Java platform editions. And the copyright owner retains the right to set the licenses, including the option to use more than one license. So the fact that Sun has open sourced the code for Java SE 6 under the GPL does not prevent it from licensing the same code, possibly with proprietary enhancements, under entirely different terms to commercial customers. Downloading the OpenJDK source code and building it into a usable binary is an interesting pastime for developers. But it is not something most corporate IT departments want to fiddle with for their mission-critical deployment platforms. And incorporating a GPL version of the JRE into a highly proprietary software ensemble like IBM WebSphere or Oracle WebLogic is probably a legal impossibility (though I believe JBoss is licensed under the so-called Lesser GPL, which permits combination with closed source software). Of course, if an independent version of the JRE was available under the much more flexible Apache license, which permits incorporation of open source code into commercial closed source variants, that would be an entirely different story. But as we have seen, this is a story that Sun has thus far seen fit to keep locked away, far from the light of day.

But wait, there's more. Someone who wants to take the GPL'd version of Sun's JDK and develop their own variant (or commercial distro) still has to contend with Sun's JCK. You might think that the JCK for a piece of open source software would itself be open source. You would be wrong! If you want to certify your version of the JDK built from Sun's OpenJDK, you have to persuade Sun to license it to you. The terms under which Sun grants such licenses are notoriously opaque. Red Hat has signed a JCK agreement with Sun for the OpenJDK in order to build an optimized version for JBoss, but the exact terms of the agreement – and the nature of any restrictions in it – are not public. However, Sun has made it clear that it is not willing to provide a JCK to developers who wish to build their own JDK from scratch, i.e. one that is not “substantially derived” from Sun's own code. (For an informative post on this topic, see Apache Harmony developer [Geir Magnusson's blog](#), and be sure to scroll down to the comments for a fierce back-and-forth between Geir and Dalibor Topic of Sun.)

So we see that promises of open source freedom can be deceiving. Despite the GPL'ing of (at least one version of) Java, Sun still has at its disposal a dense array of legal devices and stratagems for exercising control over Java. And all of this machinery is about to be inherited by Oracle. How Oracle intends to use it and to what ends, whether Oracle intends to change the machinery in any way – all such questions are complete unknowns at this point. It is worth noting though that Larry Ellison did toss off a casual endorsement of Google's Android during an impromptu appearance at the recent JavaOne conference in San Francisco. Does this mean that Oracle will lend a more sympathetic ear to Apache's desire to get a JCK for Harmony (used in Android)? Nobody knows.

A common argument is that once a piece of software has been open sourced it is therefore forever free of whatever nefarious ambitions its copyright owners may harbor. If the Java community doesn't like the way Oracle is handling Java, so this argument goes, then the dissidents can just fork the open source code and go their own merry way, leaving Oracle to stew in its own proprietary juices. But it should be clear from all of the preceding that the mechanisms of intellectual property transmission in the JCP and Sun-Oracle's control of the entirely non-open TCK licensing process reveal this notion to be little more than utopian fantasy.

We might also point out that the JCP has already seen at least one major revolt by open source Java developers, and by all accounts has successfully co-opted the dissidents back into the fold of official Java. The story can be told quite simply. When the JCP launched the IBM-developed Enterprise Java Bean (EJB) spec for server-side Java components a decade ago, many developers found the standard

## Can Oracle be trusted with Java?

extremely cumbersome to use. Within a few years, a revolt was brewing. Inventive Java developers improvised their own solutions to the problems EJB sought to address. These innovations still required a Java virtual machine to run on, of course, and so were not entirely free of Sun's IP, but they didn't need the vast panoply of libraries so painstakingly standardized by the JCP. Java guru Rod Johnson launched the Spring Framework, which pioneered new fangled concepts such as dependency injection and annotations, while another group of programmers developed Hibernate for object persistence.

By 2003 a distinct air of panic hung over the temples of the JCP, as working Java developers began deserting the approved J2EE standards in droves. But the guardians of official Java still had the vast developer resources and commercial might of IBM, BEA, Oracle and Sun behind them. They organized a vigorous counterattack, which consisted largely of adopting the opposition's ideas and inviting its leaders to sit on their councils. Rod Johnson even became a member of the JCP's inner sanctum, the Executive Committee. Java SE 5 released in 2006 contained a completely revised EJB 3.0 specification that borrowed the techniques of the Java rebels. Meanwhile, the heavyweight Java application server vendors went out of their way to signal their willingness to collaborate with the upstarts, whose mindshare it must be remembered greatly exceeded their actual monetary share of the market. The morale of the story is that a true open source fork of Java is not possible for legal and technical reasons, and could not in any case hope to take serious revenue share from the dominant vendors of official Java. Even today, more than two years into Sun's cautious tactical experiment with open source Java, there is no real certainty in the community about the future of the OpenJDK, and there is no guarantee that Sun (or now Oracle) will keep future editions of the JDK open (see for example the controversy over the [mysteriously delayed version 7 of Java SE](#), whose announcement conspicuously failed to occur at the recent JavaOne show.)

## Why Oracle and IBM won't blow up the JCP

By now it should be clear that the Java Community Process, although controlled by Sun, could not exist without the willing participation of the Java world's most important players, who include IBM, Oracle (together with its prey BEA, once the leading vendor of Java application servers), Google, and the cream of the world's mobile telephony industry. The JCP is a strategic extension of Sun, but it is also a coalition of Java vendors who have tacitly agreed to limit competition within the group in order to better face the competition from without. One witty blogger has even dubbed the JCP the "[Java Cartel Process](#)," and this description is far from unjust.

What will happen when Oracle assumes Sun's role as the strategic power behind the throne in the JCP? Will conflict erupt with the other tenor of the Java world, IBM, which has long chafed under Sun's domination of the JCP and may harbor a grudge over its humiliation at Ellison's hands in the takeover battle for Sun? Will Oracle try to manipulate the Java standards-making process or, more plausibly, leverage its control over Java IP and TCK licenses to gain advantage for itself? Indeed, while Oracle and IBM maintain an eerie public silence over the future of the JCP, other industry observers are [already speculating](#) that the deal is "the beginning of the slow, inexorable death of Java."

My own view is that predictions of the death of Java are premature, and reflect a fundamental misunderstanding of the economics of the Java ecosystem. Yes, Oracle will certainly seek to serve its own ends with the JCP. But no, Larry Ellison is not likely to allow war to break out with IBM, or with any other strategic rival in the JCP. Nor will IBM set about lobbing stones inside a structure that is the organizational equivalent of a glass house. Why not? Because to do so would risk harming the goose that lays the golden Java eggs, and neither Oracle nor IBM have any incentive to do that. After all, the JCP is a cartel that has survived many crises through the years, and the hallmark of a successful cartel is the understanding by its members that they have much to lose from anarchy and more to gain from harmony.

## Can Oracle be trusted with Java?

The principal thing that Oracle and IBM have to gain from cooperation within the JCP is easy to name. It's money. According to IDC, Oracle and IBM together took home more than 80% of the nearly \$3 billion in revenue generated by the sale of Java application servers in 2007 (the most recent year for which data is available). Oracle together with Sun accounted for 50% of the market, while IBM's share was 33%. Moreover, the Java application server market by itself is only the largest component in a broader Java middleware market, which includes a wide range of related integration middleware, much of it also based on JCP standards. IDC sizes this larger Java market at over \$5 billion, and the market share figures reveal that it too is dominated by IBM and Oracle.

When the Java "cartel" first evolved its primary target was Microsoft. But while the head-to-head against the heretics in Redmond remains intense, over the years Java has also found itself locked in competition with new rivals, namely the teeming crowd of open source scripting languages and their associated application servers that dominate the web. On the one side, Java must prevent the rise of .Net in the enterprise; on the other, it must struggle for mastery of the Internet with PHP, Python, and Ruby. The open source languages of course generate only a tiny fraction of the revenue that Java does, since many of their users don't pay for them. But they pose a serious threat to Java nonetheless, because they steal away its mindshare, especially among the young.

The IDC numbers also show that the narrowly defined Java application server market is more than six times the size of Microsoft's comparable .Net business. As noted, the much talked about open source application servers, such as those based on PHP or Ruby or Rod Johnson's Java-based Spring Framework, generate such small direct revenue streams that they scarcely register in IDC's accounting. Most developers working with these platforms earn their living from consulting and support fees rather than the sale of software licenses. In short, while open source captures a disproportionate share of media attention, and Microsoft retains high mindshare among PC-oriented developers, the server-side Java market has evolved into a de facto duopoly of Oracle and IBM. That's reason enough for both of them not to blow up the JCP.

Today we can describe the broad landscape of the application server market – all languages confounded – as a vast low revenue plain inhabited by Microsoft and the web scripting languages, dominated by the towering high revenue peak of Java. It's a good bet that Oracle and IBM will do everything in their power to make sure that nothing disrupts this financial topography so favorable to Java. In practice, that likely means a continuation of the secret backroom deals and public obfuscation that have long characterized the conduct of the Java world's leaders. The only difference is that, with pesky Sun now out of the way and no longer able to demand a degree of control not justified by its actual weight in the market, official Java and the JCP can now at last be ruled in a cozy and lucrative tête-à-tête by the two heavyweights, Oracle and IBM.

### **Does Java still have a chance for freedom?**

If there is any justice in the software world, however, there is a chance – a small but perhaps not impossibly small chance – that the Java story could have a happier outcome. What might that outcome look like? Why, freedom, of course. Freedom for Java and freedom for the millions of developers who use it, freedom to innovate and compete without paying tribute to the guardians of official Java. As it happens, the road to liberating Java and the JCP was mapped out by none other than Oracle itself. In December 2007 Oracle submitted the following resolution to the JCP Executive Committee (cited in [Stephen Colebourne's blog](#)):

## Can Oracle be trusted with Java?

"It is the sense of the Executive Committee that the JCP become an open independent vendor-neutral Standards Organization where all members participate on a level playing field with the following characteristics:

- \* members fund development and management expenses
- \* a legal entity with by-laws, governing body, membership, etc.
- \* a new, simplified IPR Policy that permits the broadest number of implementations
- \* stringent compatibility requirements
- \* dedicated to promoting the Java programming model

Furthermore, the EC shall put a plan in place to make such transition as soon as practical with minimal disruption to the Java Community."

Is there any chance that Oracle will honor the idealism and sense of fair-play exemplified in this remarkable text? Only time will tell. For the moment it's hard to see what could constrain Oracle to live up to these noble ideas, other than a little prodding from the Department of Justice. Reports in the [financial press](#) suggest that the DoJ is about to give Oracle a pass on the acquisition of Java. It would be a shame if they did so without first taking a close look at the possibility of requiring Oracle to divest its control of the JCP. Liberating Java from the clutches of a small cartel of major vendors would undoubtedly weaken the pricing power of these vendors. But it would trigger a wave of technical and marketing innovation that would bring substantial benefits to the hundreds of thousands of developers, the millions of enterprises and the billions of consumers who use Java every day.

By Jeff Gould, Peerstone Research

[jeffgould@peerstone.com](mailto:jeffgould@peerstone.com)