Java Turns 25

What’s next for the original “write once, run anywhere” programming language?

Publication Date: 04 May 2020 | Product code: INT002-000282

Bradley Shimmin

Summary

Catalyst

On May 23 this year, the Java programming language will turn 25 years old. Created by James Gosling at Sun Microsystems in 1995 as a means of supporting an anticipated market for interactive televisions and other “living room” devices, Java has since evolved into an overwhelmingly dominant programming powerhouse, influencing nearly every facet of corporate and consumer software development.

What’s left, then, for Java and its current caretaker, Oracle Corp? It has been a full 10 years since the company acquired Sun Microsystems and with that the Java programming language. As it turns out, there is plenty of opportunity left on the table, assuming Oracle wishes the language to retain its leadership position and not fall prey to old rivals and emerging disruptors alike. In this report, Omdia takes a close look at the Java programming language on its 25th birthday and discusses how this important language will need to evolve in the face of an enterprise market that is rapidly moving beyond the very notions that brought Java into existence and have allowed it to remain relevant to this day.

Omdia view

Every year new programming languages come into existence and vie for the attention of software developers. Market estimates from various sources put the current count of available languages at just over 700! And like snowflakes, no two of these 700-plus programming languages look or behave exactly alike. While most adhere to some basic agreements revolving around language syntax and approach (functional, procedural, or object oriented), each language emerges with a unique approach to anticipated market demands or as yet unresolved technical challenges. Over the long term, the success or failure of each new language depends on how well it can evolve to meet those demands and challenges year after year, customer after customer, and developer after developer.

So far, the object-oriented Java programming language has fared quite well in this regard. Even though the initial demand for set-top boxes sputtered back in 1995, when Java was first introduced, the language has shown time and again that its unique approach to platform agnosticism via the Java Virtual Machine (JVM) can evolve to successfully tackle unanticipated market turns while supporting a wide array of enterprise-class workloads running on everything from tiny embedded systems to hyperscale cloud platforms.

However, nothing remains a certainty – the market is currently undergoing a tectonic shift as enterprise developers look for tools well suited to the unique demands found within machine learning (ML), Internet of Things (IoT), and cloud-first software development. Because Oracle functions as both a caretaker and

heavy consumer of the Java language itself, developing most of its software in Java, the company has a lot to lose.

In Omdia's opinion, the work Oracle began a few years ago in moving to a six-month update cycle and introducing a new level of modularity, puts the vendor in good stead with its constituency of approximately 12 million developers. However, Oracle and the Java programming language need an ongoing series of innovative, must-have, and "delightful" features that make the language even more user friendly and cloud capable. These will keep existing Java developers happy while steering potential Java developers away from newer languages like Rust and Kotlin.

Key messages

- Oracle has been working over the past two years to modernize Java through speed and modularity.
- Now on a six-month update cycle, the newest iteration of Java (version 14) emphasizes flexibility, efficiency, and above all geniality.
- Via several projects driven by the Java community and supported by Oracle, Java is slowly evolving into a cloud-first language.
- Despite a rising tide of competitors, Java appears ready for the next 25 years thanks to its enduring flexibility, vibrant ecosystem, and Oracle's continued efforts as its caretaker.

Vendor recommendations

- **Target data scientists .**With many modern languages (Go, Scala, etc.) targeting current challenges faced by data scientists in bringing ML workloads to production, Oracle should accelerate its efforts to help bring enterprise scale, security, and speed to ML workloads in production. Such efforts will dovetail effectively with the company's related work to store ML artifacts and resources within Oracle Autonomous Database Warehouse.
- **Emphasize community .**The long-term success of Java will depend not just on careful stewardship from Oracle but also from the continued engagement among the more than 1,000 participants developing the technical specifications for Java, and the more than 12 million programmers working in Java itself. Even though Oracle drives the innovation that eventually turns into JDK Enhanced Proposals (JEPs) and broader projects such as Valhalla and Loom, the company must push its community to the forefront of innovation, allowing for and supporting as many user-driven language development efforts as possible.
- **Keep Java free and open .**Since Sun Microsystems first released OpenJDK as an open source project in 2007, this important, community-owned technology has served as more than a mere reference platform for Oracle's implementation of Java, Java Standard Edition (Java SE). OpenJDK

in many ways “is” Java in terms of developer perception. For this reason, Oracle must work to maintain developer trust in its willingness to keep Oracle both free and open. Oracle can best build trust by communicating transparently with users regarding the evolution of Java, something the vendor has done well. But it must also steer clear of litigation (as with its ongoing row with Google), as these actions send a message that Oracle prioritizes ownership over innovation.

- **Tread carefully with subscriptions.** During the summer of 2018, Oracle transitioned from its previous high-cost perpetual commercial license for Java to a low-cost subscription-based model, covering both the Java binaries and support. This puts the onus of maintaining developer confidence squarely on Oracle’s shoulders, as the vendor will need to prove the value of Java as well as its ability to support customers with every six-month update. The best way for Oracle to do this is to continue investing in and driving innovation within Java, bringing new features to market rapidly and, most importantly, by delivering predictable quarterly security updates, backed up by the company’s enterprise-class support infrastructure.
- **Bring the “wow.”** As demonstrated by the introduction of features such as multiline text blocks and higher-level switch statements, Oracle has prioritized the “developer experience,” making Java not just more efficient but also more delightful to use. This may not sound that important, but it can be argued that the reason for the continued dominance of venerable languages such as Python and the rapid emergence of TypeScript is programmer affinity. With Python, for example, this revolves around the language’s numerous inventive features such as the recently introduced walrus operator (e.g., “:=”) and long-running list comprehension capability. For TypeScript, developer affection stems from its ability to retain the simplicity of JavaScript while fixing its inherent limitations. Oracle should, therefore, prioritize the introduction of features that capture the heart as well as the mind.

Technology overview

Reimagining Java, or how to turn an ocean liner into a speedboat

Two unshakable realities exist within the enterprise technology marketplace. First, at least one pithy quote exists for every market trend. Second, for every market trend, there exists at least two “preferred” programming languages. To illustrate, back in 2011, Marc Andreessen famously said that “software will eat the world,” presaging the rise of Amazon as the world’s biggest bookseller, where even its physical bookshelves themselves turned into software with the advent of the Kindle e-reader. It is interesting to note, therefore, that the

Android-based Kindle Fire OS itself was written in three languages: C (for the core of the skinned Linux-based platform), C++ (for various services), and Java (for the user experience). Apparently, when software eats the world, it eats itself as well, at least when it comes to programming languages.

Why are there three programming languages for Kindle Fire OS? Many reasons drive organizations to co-mingle languages within a given project, involving the reuse of legacy code, language-specific capabilities, developer familiarity, etc. From the developer's perspective, language selection comes down to affinity, familiarity, skill, etc. The point is that there is always a choice. For a company like Oracle, however, which officially owns what is arguably the world's most popular programming language, there is no ostensible choice. It's Java or bust. Oracle needs Java to succeed in the long term because doing so will keep the vendor in close proximity to one of the world's largest developer ecosystems, totaling more than 12 million in all.

More than that, the success of Java and Oracle are both inexorably intertwined, as Oracle has rewritten a great deal of its internal software in Java and made Java a key component of Oracle Database and Oracle Gen 2 Cloud Infrastructure. The same holds true for millions of enterprises that rely on Java to drive mission-critical software. For Oracle, this extends to the company's push to the cloud. Thanks to numerous language-specific tools and ample supportive services, the Oracle Gen 2 Cloud Infrastructure platform stands as the best place to run enterprise Java code – and that is code not running on the 45 billion Java Virtual Machines (JVMs) currently executing Java code around the globe.

With so much at stake, it's tempting to anticipate that Oracle would look to make a big splash in celebrating the 25th anniversary of Java this coming May 23. On the contrary, Oracle's most recent release, Java 14, which will forever be associated with this important milestone, does not try to take on the world in one go. Rather, Omdia sees in that release a gentle acceleration and further realization of some very big gambles made by Oracle back in 2017 and 2018. In short, during those two years, Oracle introduced the concept of modules (Java 9) and moved the language to a six-month release cycle (Java 10). So far, the Java community has responded quite positively to these changes, as they seek first and foremost to make Java a more flexible language.

With the introduction of Java 9 Platform Module System (JPMS), Oracle converted Java from a monolithic one-size-fits-all solution to a "composable" platform where discrete functionality could be managed independently from the main Java codebase, making the language instantly more manageable at scale and nimbler in gaining new functionality. Relatedly, with release 10, Oracle abandoned a monolithic, feature-driven approach to language updates where major releases appeared every two years on average.

Together, these two large-scale changes enabled Java to evolve beyond its roots as a "write once, run anywhere (WORA)" programming language to become a "write once, run **anywhen**" programming language. Both technical partners and enterprise developers alike can now rely on a steadier stream of updates with

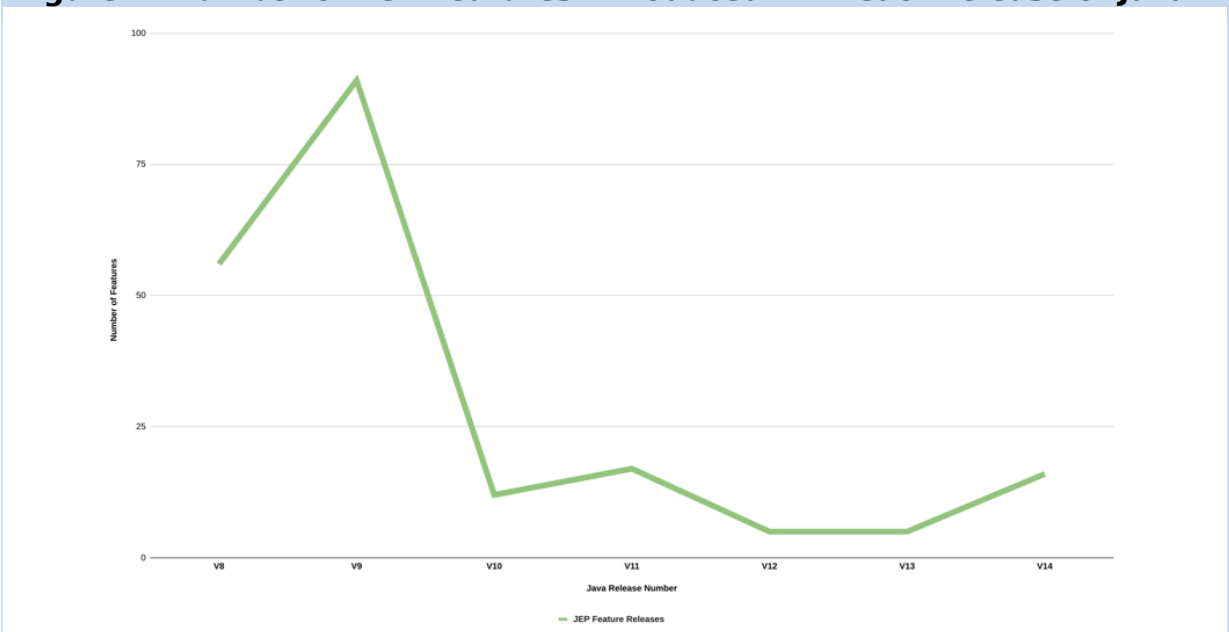
much less worry than major, monolithic changes might break running software. Of course, this also means that Oracle itself cannot look very far ahead in creating a roadmap for Java, as many new features might fall in or out of a given, six-month update. Even so, a more modular and more interactive language allows Oracle to stay in closer proximity to evolving market trends and customer demands.

It is also important to note a third major gamble by Oracle: the introduction of a straightforward annual licensing and support plan, Oracle Java SE Subscription. Introduced 18 months ago, this new plan grants full 24/7 support on a per-desktop, per-server annual subscription basis. That is just for Java on its own. For customers of Oracle Gen 2 Cloud Infrastructure platform or Oracle Fusion applications, this full support program is included as a part of those subscriptions. Importantly, licensed and supported updates for this program emerge in lockstep with Oracle's six-month release cycle as well, forming a very simple means for enterprise customers to take advantage of new features as they reach maturity.

What makes Java 14 special?

What does this mean for the latest incarnation of Java? Perhaps surprisingly, there are only a handful of major features within Java 14, as compared with earlier, more monolithic editions. The last multi-year release, version 9, contained 91 JEPs. Note that JEPs are simply collections of "non-trivial" changes, as defined by Oracle and the Java development community at large. Comparatively, all subsequent Java releases, which come out twice yearly, contain an average of only 10 JEPs (see Figure 1).

Figure 1: Number of new features introduced with each release of Java



Source: Oracle

Broadly speaking, since moving to a six-month update cycle, Oracle has somewhat narrowed the scope of features introduced with each release, delivering only sixty JEPs after version 9. However, more features delivered as a huge clump does not guarantee timeliness or innovation. With a rolling set of delivery opportunities every six months, however, Oracle expects to deliver more topical features more frequently without sacrificing the quality, as new features are still vetted by both Oracle and the Java development community at large. The JEP process itself usually begins with Oracle solving a discrete problem internally and then taking that to the broader OpenJDK community as a JEP. Each JEP will join the main code tree of the Java standard in preview or final form at the most opportune time – neither being rushed into nor languishing between infrequent, monolithic releases.

With this in mind, the new set of 16 JEP features delivered as a part of release 14 seem more evolutionary than revolutionary. New features include the deprecation of support for Solaris and Sparc, the ability to use Z Garbage Collector (ZGC) on MacOS and Windows platforms, and the addition of NUMA-Aware Memory Allocation for G1 garbage collection. These all concern, quite literally, the maintenance of the Java platform. On their own, these won't keep Java at the forefront of enterprise software development.

Oracle and Java both need to regularly win over the Java developer community to keep existing developers happy and productive while encouraging would-be Java developers to invest the time necessary to learn Java. On the whole, release 14 endeavors to answer this challenge by introducing features that make Java more flexible, efficient, and above all friendly. Here's a short rundown of four standard and preview features released with Java 14 that fill this bill. Note that a JEP preview feature denotes functionality that may evolve from release to release. Standard JEP features are considered a stable part of the Java specification going forward.

- **JEP 361: Switch Expressions .** This capability first showed up in Java 12 and is now a standard feature. It extends the switch expression, allowing it to function as either a statement or as an expression. Historically, the Java switch expression, which closely follows C and C++ switch expressions, works best on very low-level comparative operations. JEP 361 frees the switch expression to better support higher-level contexts with less overhead. In the long term this is important, because it paves the way for the following feature, JEP 305.
- **JEP 305: Pattern Matching for instanceof .** Included as a preview feature in Java 14, this improvement to the switch statement lets developers look for and test against various object structures in a more concise and less error-prone manner than previously. Other modern languages use this same sort of capability, most notably Java rivals Haskell and C#. Once this enhancement becomes a standard component, Oracle will work on more advanced pattern-matching constructs, using switch expressions and statements.

- **JEP 359: Records .** Also in preview, this JEP takes aim at a common complaint among Java developers, namely that the language is too verbose, particularly when it comes to handling classes that simply carry data. Python has tuples, and Scala has case classes, for example, which simplify this sort of requirement. Not looking to simply copy these approaches, Oracle is instead looking to introduce similar levels of simplicity with a class type as a transparent holder for data that does not demand a high degree of boilerplate information.
- **JEP 368: Text Blocks .** Available as a preview (for the second time), this feature seems innocuous in allowing developers to reference text that spans more than one line without having to make use of escape characters. Text Blocks first and foremost makes Java code more readable but, more importantly, it lets developers reference external code blocks (HTML, XML, SQL, or JSON, etc.) without having to break up code using highly illegible and error-inducing escape codes. This makes Java a more friendly player among cloud-first software, where developers routinely work with multi-line text blocks in messaging between applications.

As you can see from this limited set of features, Oracle and the Java development community are working on a number of related threads across numerous release cycles. Often, new capabilities will morph or even move between JEPs, as was the case with JEP 368 (Text Blocks). This was initially proposed as a part of JEP 355, which was previewed in the previous Java release. Subsequent feedback from Java 13 users prompted the reassessment and extension of this preview for release with Java 14.

What’s next for the Java programming language?

Looking forward to Java 15 and immediately beyond, Oracle will continue with this same mix of enhancements, balancing those that keep Java running and relevant as an enterprise-grade development platform with those that delight and empower Java developers. For example, slated for release 15, JEP 372 will finally remove the Nashorn JavaScript script engine and APIs, which had proven difficult to maintain over time. With JEP 371 (also slated for version 15), Oracle will introduce hidden classes, which are classes that cannot be accessed directly by the bytecode found in other classes. This will help Java to better handle ideas like code generated at runtime as found in Lambda functions and serverless computing.

Such enhancements solidify Oracle’s role as a conscientious caretaker of this important language, which is one variable in a complex equation. A more impactful and still undecided (possibly confounding) variable concerns the “wow” factor. This is where JEP 368 (Text Blocks, which were mentioned above) comes into play as a singular example of exactly what Oracle needs to make Java successful over the long term – that is, appeal to developers working in cloud-first environments where apps communicate via JSON, and users interact with HTML, not Java.

Fortunately, Oracle is pushing Java in this direction, as evidenced by an as yet unscheduled JEP 369, which will effectively move the open source rendition of Java (OpenJDK) to the popular version control system, GitHub. The company has a lot more of this sort of contemporary thinking in mind, housed within a series of incubator-style projects, each designed to tackle specific opportunities across multiple JEPs. Below is a short look into the current status of a few key projects:

- **Project Panama .** Built to create better connections between the Java JVM and external, non-Java programming languages, Project Panama brings together a number of efforts headlined by native function calls for C and C++ from within the JVM. This project recently saw an important supportive element enter production with the Java 14 release, namely Foreign-Memory Access API enhancement (JEP 370).
- **Project Valhalla .** Initiated back in 2014, this ambitious project focuses on bringing Java into closer alignment with modern hardware, particularly in how the language accesses data across cache and memory stores. This is a key notion when considering Java as a platform for ML endeavors, where hardware optimization is crucial. In support of this project, in August 2019, Oracle made available to developers an early access build of the language that included key Valhalla capabilities, namely L-World Inline Types. The vendor intends to bring more preview functionality to the Java community throughout 2020 as well.
- **Project Loom .** This project changes the way Java handles concurrent programming via threads by introducing the notion of fibers. This idea, which lets a thread basically wait for a number of asynchronous responses without consuming a great deal of system resources, will help Java play better with large-scale cloud-first projects where rival languages like Kotlin shine. As with Project Valhalla, there are early access developer builds of Loom available for testing now.
- **Project Amber .** One of the most active Java community projects in terms of delivering JEPs into production, Project Amber aims to make Java developers more productive. Targeting the way developers write code with enhancements such as Switch Expressions (JEP 361), Records (JEP 359), Pattern Matching for instanceof (JEP 375), and Text Blocks (JEP 368), Amber carries the most weight in terms of both delighting existing Java developers and enticing new recruits to invest the energy necessary to learn a new programming language.

Clearly, as the lead contributor to, and steward of, the Java language, Oracle is playing the long game when it comes to initiating and supporting forward-looking projects. And yet, the biggest danger and challenge for Oracle rests not so much within the evolution of the Java language itself, but rather in the way programming languages in general have evolved since the early 1990s, when Mr Gosling and his colleagues began working on the Java language.

Programming languages are not binary decisions

Because Java functions as both a programming language and as an execution environment (via the JVM), it has actually helped to spawn new programming languages outside of Java itself. As the foundation for WORA development, the JVM has become a ready home to numerous languages that both augment and compete with Java. The most notable of these is Kotlin. A recent creation (circa 2010), Kotlin is a statically typed language that can be compiled to run as either JavaScript, natively as its own binary, or within a JVM (with 100% Java compatibility).

This affinity for the JVM, coupled with a number of cloud-first, productivity-friendly features like Lambda expressions + Inline functions, has helped Kotlin quickly gain developer support, particularly among Google Android developers. This in turn has led Google in 2019 to nominate Kotlin as its preferred language for Android app developers. Java developers working in Google Android Studio can work with both Java and Kotlin code natively, mixing the two and even converting from one language to the other.

For Android developers in particular, this sets the bar of entry into Kotlin and exit from Java pretty low, as they can decide on a project-by-project basis how best to write and run their software. If Java developers decide that they would rather write in Kotlin because it purportedly requires 20% less code, employs a time-saving feature like operator overloading, or better supports the idea of functional programming, then there is very little standing in their way.

Even so, Oracle sees languages like Kotlin as vital to its overall success. First, anything that enhances the value of Java's JVM as a deployment platform helps ensure the long-term viability of both Java and JVM alike. Second, Oracle is actively learning from projects like Kotlin now it can improve both language and platform alike. In a way, Kotlin itself can be seen as a low-risk, innovation sandbox, where innovations that might find their way into Java can be explored more freely among the Java community at large.

There is a broader, more existential threat that Oracle must face as well. As demonstrated by Kotlin, it does not take 25 years for a language to shift the hearts and minds of the developer community at large. Within the well-regarded TIOBE Programming Community index, which measures the popularity of programming languages, Java is currently ranked as the world's number one programming language. Note that Kotlin shows up as a meager 30th on this same list. Comparatively, the top four most beloved languages, as defined by Stack Overflow's 2019 survey of more than 90,000 technology practitioners, were (in order): Rust, Python, TypeScript, and Kotlin. None of these languages, save Python, existed before 2011.

Do these studies paint a picture of Java as an incumbent leader under threat? Are Java developers likely to jump ship to work in more "beloved" languages like Rust, TypeScript, or Python? On any given day in response to a specific given project, developers might choose Rust over Java if they require complete

memory safety in a very small footprint. They might choose TypeScript over Java if they are dedicated to the Microsoft developer ecosystem and hate JavaScript but still need to build JavaScript-type applications. They might choose Python over Java if they are data scientists looking to quickly build and iterate an ML model. Actually, for Python developers, Java would not come up as an everyday competitor, except as an alternative to the JVM-based Scala language as a means of deploying a very highly performant inferencing engine for use cases like fraud detection.

Does this mean Java needs to do more to compete with Scala in order to win over data scientists? That would not hurt. As mentioned above, Oracle is working actively on modifications to the Java language that will make it more appealing within the use cases where Python developers thrive. But that is not the most important thing. For any given situation, enterprise developers will select the language that best meets the needs of the current project at hand, answering three important questions:

- Do the developers have the necessary experience and/or supportive tools at hand (libraries, frameworks, etc.) to build that project in time?
- Will the language provide the required scale, performance, and security?
- Will the language run as desired on the target platforms (mobile, web, etc.) and access the desired resources (database, hardware, etc.)?

For any given project in any given enterprise, out of the approximately 700 available programming languages, maybe only a handful will meet all of these needs – and that list will change from project to project. As mentioned at the outset, for any given need there are at least two languages that will get the job done. But that is actually the very crux of Java's success. What has allowed Java to thrive for so many years and to remain in a dominant market position can be summarized in one word: flexibility.

Java's main calling card has been, and continues to be, its ability to accommodate the task at hand to say "yes" to those three questions. This flexibility is echoed in the varied and influential list of top Java community contributors, which includes Red Hat, SAP, Tencent, Google, Intel, IBM, and Amazon. So long as Oracle keeps Java freely available across more than 45 billion active JVMs, nurtures its community of more than 12 million developers, and steers Java in lockstep with the enterprise marketplace, Java will thrive.

For now, that means evolving Java to play more effectively with cloud-first methodologies, to take better advantage of new hardware architectures, and to help developers get more done with less. That will require Oracle to embrace opportunities like ML by focusing on aspects such as concurrency and threading (where such scripting languages as Python fall down) and tackling cloud-first communications by playing nice with asynchronous communications via JSON.

Tomorrow is likely to present a new set of challenges – perhaps even the challenge to move beyond Java itself just as TypeScript did as a superset of JavaScript. Perhaps Java will push Scala out of the way as the preferred means of

analyzing data at scale among data scientists. Perhaps we could even see Java emerge as a preferred foundation for low-code/no-code development where programming takes a back seat to drag-and-drop app assembly. All options are open. Until then, it is safe to say that Java and the global Java development community are in good hands with Oracle.

Appendix

Methodology

This report was written drawing on briefings, customer events, and industry events with decision-makers, technology and IT services vendors, and end users across a number of geographies. This is combined with desk research and Omdia's ICT Enterprise Insights.

Further reading

2020 Trends to Watch: Cloud Computing, INT003-000402 (October 2019)

2020 Trends to Watch: Cloud-Native Development, INT003-000416 (December 2019)

2020 Trends to Watch: Data Center Technologies, INT003-000405 (November 2019)

"Red Hat Quarkus is making Java relevant for the Kubernetes age," INT003-000363 (May 2019)

Author

Bradley Shimmin, Distinguished Analyst, Data Management and Analytics

askananalyst@omdia.com

Citation Policy

Request external citation and usage of Omdia research and data via citations@omdia.com.

Omdia Consulting

We hope that this analysis will help you make informed and imaginative business decisions. If you have further requirements, Omdia's consulting team may be able to help you. For more information about Omdia's consulting capabilities, please contact us directly at consulting@omdia.com.

Copyright notice and disclaimer

CONTACT US

omdia.com

askananalyst@omdia.com