ORACLE
JAVA EMBEDDED

Developing Intelligent Connected Devices for the Internet of Things

Six Reasons to Consider the Java Platform

ORACLE WHITE PAPER | JUNE 2015

ORACLE

Table of Contents

Java—Much More Than a Programming Language

Java is not just a programming language—it is also a platform that includes a well-known set of open source and commercial tools, an extensive ecosystem of experienced developers, and a standardized application deployment platform. Because it is flexible, scalable, cost-effective, and reduces risk, Java helps modern digital businesses to simplify development, reduce overall development costs, and accelerate time to market.

The Java Virtual Machine is adapted to run on a wide range of hardware, processors, and operating systems, which means that a Java application running on one computer or device can be moved to another with virtually no changes. This capability hasn't been achieved by any other platform, including Microsoft Windows, Android, or native code.

As a result, Java and solution developers can easily change hardware to meet other design criteria with confidence that their Java software will run on the new platform. The Java Virtual Machine abstracts hardware details, such as peripherals and their associated driver software, from the Java developer. Developers write to the peripheral's Java API and the Java Virtual Machine does the rest.

Two other unique Java aspects include a comprehensive, standard set of class libraries implemented across platforms, and the Java Community Process (JCP) that extends the Java platform. Many of the tools, class libraries, and enhancements to the Java language are products of the JCP, which is described in more detail in the section "The Java Community Process."

Six Reasons to Consider the Java Platform

The Internet of Things (IoT) is pushing corporations across industries toward change. Many enterprises want to leverage connectivity and data from remote systems that will help transform and grow their business. With mobile and wireless devices that connect to the enterprise, associated end-to-end communication needs, and data being gathered from myriad sources, the complexity of designing an Internet of Things system can be daunting.

The hardware and OS agnostic Java platform is perfect for the Internet of Things.

But the Java platform can help minimize that complexity. In this paper, we'll discuss six reasons why embedded developers should consider the Java platform:

- **The Java community and ecosystem**—Java operates within an inclusive, transparent, and highly involved community that provides the resources and best practices developers need in their approach to platform design.
- **Platform scalability**—Because Java is widely scalable—which is an important requirement for the Internet of Things—developers can enhance system dynamics from the sensor to the cloud.
- **Platform security—**Java provides inherent security via its sandbox execution model and included security frameworks.
- **Simplicity of development and maintenance**—The multiplatform Java Virtual Machine runtime comes with a large set of standard capabilities that help simplify application development.
- **Development tools**—Effective tools and libraries help maintain your system, such as seamless version control of the programming language and application updates independent of the platform.
- **Business value—**Java helps increase developer productivity while reducing license and maintenance costs.

These reasons add up to a software environment that offers cross-enterprise and cross-machine application capabilities from embedded devices to corporate servers. A wide range of general-purpose and specialized development tools and a community of engaged users and experts help ensure Java remains manageable, scalable, secure, and accessible. Read on to explore the advantages of the Java ecosystem for your business and technology needs.

The Java Community and Ecosystem

With 2015 being the 20th anniversary for Java, it remains a stable, mature, and high-performing platform, traditionally ranking at the top of the TIOBE and IEEE indexes of software platforms (see Figure 1). Java relies on a community-driven process to mature and evolve, beyond the control of one single corporate entity. As a result, Java has amassed the world's largest global community of developers who can use their knowledge and expertise across both the enterprise and embedded problem domains.

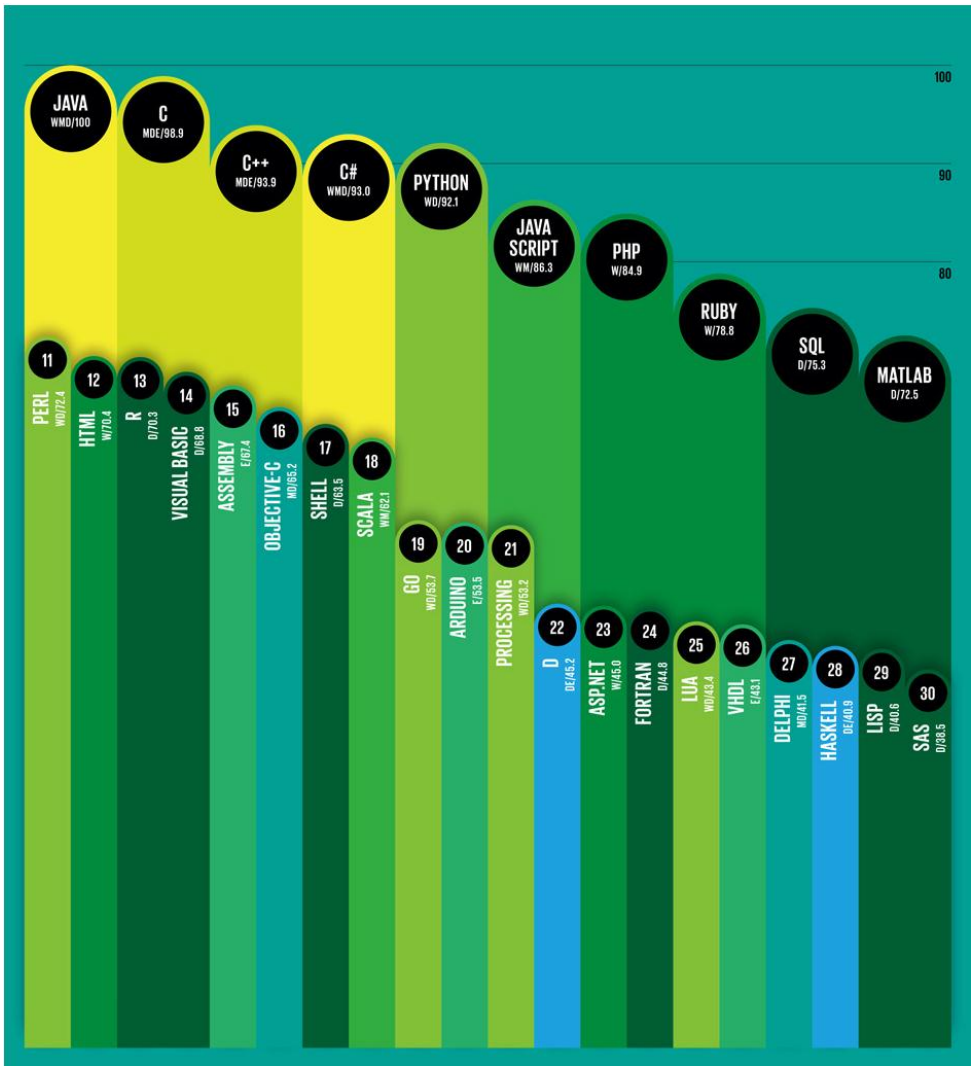

Figure 1. IEEE Spectrum's 2014 top-ranked programming languages

While Java evolves via the Java Community Process, both Microsoft and Google exercise corporate control over the evolution of their platforms. There have been cases where Microsoft has deprecated technologies (i.e. Silverlight) leaving developers to find their own migration path. In contrast, since the Java Virtual Machine is compatible on different versions of Windows, Java applications will run equally well across each of them.

The Java Community Process

The Java Community Process (JCP) was established in 1998 to give the growing Java community a voice in producing standard technical specifications and to allow Java to evolve with the changing technology requirements of the community. The JCP's open yet formal practices allow the entire community to participate—a contrast to proprietary software platforms where evolution is done solely through a usually closed, corporate-driven development process.

The JCP allows for technical reviews, evaluations, and responses that go beyond a single organization or group. Java specifications are written and tested by technically qualified individuals from Expert Groups—members with technical knowledge associated with a specification request and who can select others to be members of their Expert Group. During the process, any JCP member can review and comment on milestone drafts of the specification.

To add new functionality to the Java platform, the Java Community Process begins with a Java Specification Request (JSR), which can be submitted by any member (a company or an individual). Members may review and comment on the progress of the specification, and the JCP Executive Committee votes to select JSRs for development and to approve final specifications. The Executive Committee is responsible for guiding the overall evolution of Java and ensuring that new specifications do not overlap or conflict with one another. The Executive Committee is elected by JCP members and represents the international Java community. Oracle is a member of the Executive Committee but does not direct it.

Once a JSR Expert Group has finished collecting public feedback, it issues a reference implementation (RI), which is a proof-of-concept implementation of the specification. In addition, the group produces a technical compatibility kit (TCK), which is a suite of tests, tools, and documentation ensuring that any future implementation will be compatible with the specification and the language itself. Finally, the specification is sent by the JSR Expert Group to the Executive Committee for final approval.

The result is that Java is well prepared to take on applications from embedded to enterprise systems without risk of becoming fragmented.

Embedded and the Internet of Things market opportunities are growing so rapidly that existing embedded engineering resources cannot keep up with its time-to-market and content creation requirements. Not only do organizations need to be more efficient, but also the existing population of embedded engineers needs to be able to scale organically to meet the new software content creation requirements. Today, only 30 percent of the world's 1 million embedded engineers hold software engineering–specific roles. In order to adapt to the new IoT development demands, organizations must look to new labor resources. The global Java community, estimated to include about 9 million developers, offers an opportunity to draw upon an increasingly relevant labor and expertise pool ("Engineering Business Value in the IoT with Java 8," VDC Research, 2014).

Although being open source, Android software releases from the Android Open Source Project (AOSP) have no identifiable technical roadmap while key features of the Android platform (such as the Play Store) are only available as proprietary components from Google. This model does little to provide stability and direction for future development and severely limits the attractiveness of the Android Open Source Project. Additionally, the Android Compatibility Program defines only three-supported device types—handheld, television, and watch—that each requires display support. This means that many embedded developers have to choose between customizing

solutions beyond the bounds intended by the AOSP or being limited to a partial solution implementation. This creates technical hurdles and additional support expense for Android application development teams

Unmatched Java Support

Java offers high-level support for both the community and the enterprise. In addition to the benefits of the JCP, Oracle Support offers programs that are scalable and provide multiple licensing and support options with additional premium management and monitoring features to help decrease operational costs for both the Java platform and Java applications.

Just as Java has proven itself ready to solve the most complex system solutions, it asserts its leadership with a fully transparent and active community model.

For example, the Oracle Java Embedded solutions—Oracle Java ME Embedded, Oracle Java SE Embedded, and Java Card—have dedicated commercial support designed for embedded developers. For ISVs, there is Oracle Java SE Advanced—specialized, 24/7 technical support that leverages the knowledge of Java experts at Oracle and which is available in more than 27 languages. Both commercial support options include access to fixes, thoroughly tested product releases, security updates, and personalized proactive support tools. Add to this Oracle's commitment to support any release of Java for no less than 11 years from release date, and developers have the best software support system in the world.

Microsoft follows close behind with 10 years of release support. However, its policy of abruptly ending platform variants with each new release effectively abandons developers working to create platforms that can evolve and scale. Some may consider this better than the alternative of Android and native development, which provide no real organizational support or commitment.

Developers directly benefit from Java's open roadmap since they can influence the technology, leverage a huge base of available code, deliver solutions quicker, and build skills that are valuable in the marketplace.

Platform Scalability

Java is designed to provide the most flexible and interconnected software environment available. Given the wide adoption of the Java language, the availability of many OS and platform-specific implementations of the Java Virtual Machine, and a community that evolves with the latest technology, developers have a true partner in Oracle. Whether -building embedded, cloud, or end-to-end solutions, Java scales to provide the interoperability developers need.

Portability exists not only across hardware and operating system configurations but also across releases of the Java language and Java Virtual Machine. Long-term backward compatibility ensures that Java applications written today will work equally well with the Java versions of tomorrow.

When it comes to embedded device development, a smaller footprint is generally better. With Oracle Java Embedded, the same application code can run on a wide array of devices and systems based on varying embedded hardware and software environments. Currently, Oracle offers small-footprint Java Virtual Machine implementations with headless configurations and memory optimizations for x86, ARM, PowerPC, and other microprocessors.

"In the past five years, the percentage of engineers using Java in the embedded market has more than doubled." — "Engineering Business Value in the IoT with Java 8," VDC Research, 2014.

Small-Footprint Java Runtime Environment

Through compression and optional file removal, Oracle Java SE Embedded can provide a reduced-footprint Java runtime environment (JRE) as required by many embedded systems. A JRE can shrink to less than 25 percent of the full Oracle Java SE Embedded JRE footprint—as low as 11 MB for Java SE 8.0 Compact Profile 1 compared to 48 MB for the full Oracle Java SE Embedded JRE. Additionally, Oracle Java SE Embedded can run on devices with as little as 16 MB of memory headless, or 32 MB headful (supporting a user interface).

Oracle Java ME Embedded, which targets smaller embedded devices, runs on resource-constrained devices including micro-controllers with as little as 128 KB of memory. Compare that to Microsoft Windows Embedded 8, which requires 1 GB of memory and 3 GB of storage (and suggests 6 GB).

Because Android isn't specifically designed to meet the challenges of embedded applications, scalability isn't available. Native C/C++ applications have some ability to scale, but their development involves a huge effort to support different platforms and processors. Both Android and native development require the developer to build solutions that scale across devices and then provide ongoing maintenance of those builds for the different targets.

Developers gain from this as Java enables the following:

- The re-use of code and solutions across different hardware and platforms
- The re-use of development skills and methods from enterprise to embedded projects
- The ability to leverage a consistent set of development tools regardless of target hardware and platform

Platform Security

With increased connectivity and the rapid expansion of the Internet of Things, security is becoming an increasingly vital consideration for embedded solutions. Java Virtual Machine is built with proven technologies that provide a secure environment to run multiple applications that are isolated from each other, the operating system, and other software components. Oracle delivers timely security patches and Java Virtual Machine updates so developers can focus on their solutions.

With Java, Security Is Built In

The Java sandbox provides a controlled execution environment that protects running applications from other OS processes and resources as well as other Java applications. It also supports data security in all three IoT domains: devices, networks, and services. The write once, run anywhere Java development model applies to security as well—developing componentized Java applications according to supported security standards means you will less likely need to build in new security measures as your hardware platform changes.

The security features of Java and its libraries include secure messaging, user authentication and authorization, device identity, data encryption, code signing, and more. To ensure the security of deployed and managed systems, Java provides a comprehensive set of industry-standard implementations, including a hardware module with certificate-based security, the Java Cryptography Extension with crypto acceleration, and near field communication (NFC) support. Secure communication is ensured with Java Secure Socket Extension (JSSE) and Java Authentication and Authorization Service (JAAS) for user, device, and data identity, and public-key cryptography standard (PKCS-11) for data encryption. In addition, the Security and Trust Services (SATSA) API offers encrypted security features and communication capabilities.

The Java Virtual Machine and standard security libraries provide security in a layer between the application and the operating system. C and C++, however, don't have built-in security so you must build all security elements into the overall solution, including the operating system.

Simplicity of Development and Maintenance

Simplifying application development can help accelerate time to market, streamline efforts of the software engineers, and enhance security. Java offers the developer a standard environment, while both Android and native development require the writing of extra code to address the underlying platform details. For Java developers, network and file operations, database access requests, and other actions that depend on the operating system are handled via standard Java API calls. Using these built-in libraries and available frameworks make Java code highly portable and easier to test. This is the foundation of the *"write once, run anywhere"* portability of Java applications, and it greatly simplifies application development.

Continual thought and effort go into the maintenance of the Java platform (for more information see call out section, "The Java Community Process"). Oracle works hard to maintain backward compatibility so that moving to a new version of Oracle Java SE Embedded doesn't invalidate application code. The result is that existing tested applications won't require modification, representing considerable savings.

In contrast, native applications written in C/C++ often require greater validation testing when upgrading the operating system or hardware environment as well as porting to different operating systems and processor configurations. Additionally, C/C++ developers must specifically manage memory, which requires greater skill, more development time, and more complex debugging, and presents a greater chance of application failures at runtime. Maintaining native code can cost significantly more than Java, requiring specialized knowledge possessed by fewer developers. Hiring native developers not only costs more, it puts your organization at risk if they leave.

Java Virtual Machine Removes Complexity

The Java Virtual Machine is unified and standardized for each hardware environment to which it is ported. Java application code is compiled to an intermediate bytecode that can be executed on any Java Virtual Machine, and for further performance this bytecode can be converted to native machine code on the fly using a just-in-time (JIT) compiler. The result is that Java applications run as fast as native applications. The value of Java is that it shields the developer from underlying platform specifics and provides a consistent application execution platform.

It's important to draw the distinction between Java and Android's "Java-like" language, Dalvik, which differs from the Java Virtual Machine in terms of bytecode compatibility and class loading. In addition, the fragmentation between versions of Android used by several major manufacturers can make Android apps incompatible across them.

The Java Virtual Machine executes on top of a selected operating system, which itself is implemented for a particular hardware environment, abstracting the details of both the OS and hardware from the developer. Android, on the other hand, includes the Linux operating system as an integral part of its package, limiting that abstraction.

In terms of native development in C/C++, you need to write the low-level code for all the functionality or integrate native components from different sources for a full solution, thereby increasing complexity and effort. A platform-agnostic language such as Java, however, shields developers from hardware details and offers a rich set of libraries for data transfer, image processing, mathematic operations, security, and more—all easily referenced via an application.

Internet of Things (IoT) projects are forcing organizations to respond to rapidly changing customer requirements while also supporting new business models and product use cases. According to a 2014 VDC Research poll of embedded developers, programmers using a language such as Java reported better schedule adherence than those using C.

Embedded application development in C/C++ requires cross-compilation and target debugging for multiple processor and operating system configurations, introducing complexity for both initial development and upgrades. Additionally, native applications in C/C++ are generally not portable to a different environment without considerable rework.

In the case of Android, an application written for a specific Dalvik version should run under that version of Android. You need to target a combination of both Android and the SDK versions that best match your target users. Additionally, if your Android application makes calls directly to the underlying version of Linux, there's no guarantee of portability.

Development Tools

The vibrant Java ecosystem offers developers a robust selection of application frameworks and development tools. Here are some examples:

- Integrated development environments (IDEs) are offered through open source community and commercial vendors.
  - BlueJ is a Java IDE designed for students.
  - Oracle JDeveloper offers enhanced integration capabilities with Oracle middleware and business intelligence products.
  - Eclipse and Oracle-sponsored NetBeans are open source project development tools.
- Build and test tools
  - Hudson is an extensible continuous integration server, which monitors the execution of repeated jobs and is used to monitor and control complex software builds.
  - Java VisualVM provides a visual interface for viewing detailed information about Java applications while they're running, and for troubleshooting and profiling these applications.
  - Java Mission Control enables you to monitor and manage Java applications without introducing the performance overhead normally associated with these types of tools. It uses data collected for normal adaptive dynamic optimization of the Java Virtual Machine.
  - Java Flight Recorder collects diagnostic and profiling data about a running Java application. This powerful tool is primarily used for profiling, black box analysis, debugging and support.
- Software frameworks
  - Open source frameworks, like those available from Oracle and others, provide advanced platforms such as GlassFish (the Java EE reference implementation) and other open source containers that you can include in your solutions.

Java Delivers a Rich Tool Chain

Java developers have a choice of toolsets from Oracle and the Java community that enable them to build applications that run across many operating systems and hardware platforms. This eliminates having to select a compiler, debugger, code profiler, and other tools for a specific platform, or having to acquire new versions of these tools or learn new processor-specific techniques when moving between hardware environments.

No Need to Develop on Target Systems

In addition, native code is most often developed on a host system—such as a PC—with a hardware and software environment that is different from that of the target embedded system. This requires the code to be cross-compiled from the host to the target environment and the construction of a tool chain that can bridge the two worlds. This is almost always a complex task involving the setup of a host development environment. Even if that environment is Eclipse, the job is far from over.

Java, on the other hand, abstracts these details from the developer so that Java development and execution is portable between hardware and OS environments. The Java Virtual Machine and Java libraries shield the developer from the OS-specific details of Windows and other operating systems.

Business Value

Software development lifecycle statistics consistently show that considerable costs occur during the maintenance phase of software development. In the May/June 2001 IEEE Software article by Robert L. Glass, “Frequently Forgotten Fundamental Facts About Software Engineering,” the 60/60 rule is introduced, which shows that maintenance typically consumes 40 percent to 80 percent of software costs (60 percent on average), and enhancements are responsible for roughly 60 percent of software maintenance costs, while error correction is about 17 percent. The Java platform reduces these costs through features inherent in the language such as the lack of pointers, automatic memory management (garbage collection), and standard high-level application programming interfaces representing code reuse, readability, and reliability.

Empirical data from many years of real-world Java projects often shows significant reductions in software development and maintenance costs of up to 80% or more compared to native solutions,

Software Licensing Models

Oracle offers its Java platform for free during the embedded development phase, which provides an early cost advantage. Once in production, Oracle offers two routes to license Java for an embedded solution: either per device or through a subscription. These options provide flexibility that caters to the differing requirements of embedded system developers.

Microsoft provides a standard development platform through its OS partnerships and requires expensive licensing structures. Although Android is open and without licensing fees, it can be costly in other ways such as lock-in and lack of portability. Although difficult to fully quantify, development dead-ends and the requirements of additional resources to manage and support Android development can be extensive once in production.

While attempting to develop a platform approach, Microsoft has decided to vacate or poorly support some technologies, such as Silverlight, leaving application developers looking for alternate solutions. And since Android is mainly targeted at the handset market, it introduces its own challenges in terms of portability.

Java Software Development Support

Oracle provides a well-documented process and roadmap for update releases and revisions of Oracle commercial implementations of the Java platform. Commercial implementations are supported by Oracle Premium Support Service, which provides 24/7 technical support, extended software lifecycle support, and access to commercial features, including Java Mission Control and Java Flight Recorder, which together offer profiling and event monitoring. Oracle also offers Java Engineering Services to address specific customer embedded device optimizations or configuration requirements.

Oracle, unlike other software platform vendors, provides documented processes and demonstrates commitment to support its products and customers.

Conclusion: Gaining an Edge on Tomorrow

The Internet of Things points to a promising future where devices will communicate with each other, end users, and the enterprise applications we use. Because the Internet of Things is at the core of how our devices and systems will be designed, platforms that enhance the connectivity of diverse devices will drive the technology market. Devices, which range from small microprocessor or sensor driven endpoints to large machines, talk to each other through intelligence driven from the Cloud.

However, interconnectivity is only a portion of the Internet of Things. The real significance is data, which helps generate insight that is used to make decisions about control. This involves sending commands back to the appropriate devices, many of which are also sources of the data. Between the data and the control is intelligence, which helps drive the insights and make the decisions. Java's portability helps distributed applications work together in a system of hardware platforms at different levels in the network hierarchy, using a common language and toolset end to end.

Oracle Java Embedded is engineered and optimized to meet the unique challenges of designing intelligent devices that unlock the business value of the Internet of Things and embedded solutions. An open, standards-based platform with an unequalled developer ecosystem, Oracle Java Embedded makes it faster and more affordable to get innovative, reliable, and secure embedded products to market and provide the long-term support needed for success.

To learn more, visit oracle.com/goto/javaembedded and oracle.com/iot.

Contact oracle at javasales_ww@oracle.com

Oracle Corporation, World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065, USA

Worldwide Inquiries
Phone: +1.650.506.7000
Fax: +1.650.506.7200

CONNECT WITH US

blogs.oracle.com/oracle
facebook.com/oracle
twitter.com/oracle
oracle.com

Hardware and Software, Engineered to Work Together