History of the Java™ programming language

On 23 May 1995, John Gage, the director of the Science Office of the Sun Microsystems along with Marc Andreesen, co-founder and executive vice president at Netscape announced to an audience of SunWorld^{TM} that Java technology wasn't a myth and that it was going to be incorporated into Netscape Navigator.[1]

At the time the total number of people working on Java was less than 30.[1] This team would shape the future in the next decade and no one had any idea as to what was in store. From running an unmanned vehicle on Mars to serving as the operating environment of most consumer electronics, e.g. cable set-top boxes, VCRs, toasters and PDAs,[2] Java has come a long way from its inception. Let's see how it all began.

Earlier programming languages

Before Java emerged as a programming language, C++ was the dominant player in the trade. The primary goal of the creators of Java was to create a language that could tackle most of the things that C++ offered while getting rid of some of the more tedious tasks that came with the earlier languages.

Computer hardware went through a performance and price revolution from 1972 to 1991. Better, faster hardware was available at ever lower prices, and the demand for big and complex software exponentially increased. To accommodate the demand, new development technologies were invented.

The C language developed in 1972 by Dennis Ritchie had taken a decade to become the most popular language amongst programmers working on PCs and similar platforms (other languages, like COBOL and FORTRAN, dominated the mainframe market). But, with time programmers found that programming in C became tedious with its structural syntax.[3] Although people attempted to solve this problem, it would be later that a new development philosophy was introduced, one named *Object-Oriented Programming* (OOP). With OOP, one can write code that can be reused later without needing to rewrite the code over and over again. In 1979, Bjarne Stroustrup developed C++, an enhancement to the C language with included OOP fundamentals and features. Sun generated revenue from Java through the selling of licenses for specialized products such as the Java Enterprise System.

The Green team

In December of 1990, a project was initiated behind closed doors with the aim to create a programming tool that could render obsolete the C and C++ programming languages. Engineer Patrick Naughton had become extremely frustrated with the state of Sun's C++ and C APIs (Application Programming Interfaces) and tools. While he was considering to move towards NeXT, he was offered a chance to work on new technology and the *Stealth Project* was started, a secret nobody but he knew.

James Gosling, architect and designer of the compiler for the Java technology

This Stealth Project was later named the *Green Project* when James Gosling and Mike Sheridan joined Patrick.[1] As the Green Project teethed, the prospects of the project started becoming clearer to the engineers working on it. No longer did it aim to create a new language far superior to the present ones, but it aimed to target devices other than the computer.

Staffed at 13 people, they began work in a small office on Sand Hill Road in Menlo Park, California. This team came to be called the *Green Team* henceforth in time. The project they underwent was chartered by Sun Microsystems to anticipate and plan for the "next wave" in computing. For the team, this meant at least one significant trend, that of the convergence of digitally controlled consumer devices and computers.[1]

Reshaping thought

The team started thinking of replacing C++ with a better version, a faster version, a responsive version. But the one thing they hadn't thought of, as of yet, was that the language they were aiming for had to be developed for an embedded system with limited resources. An **embedded system** is a computer system scaled to a minimalistic interface demanding only a few functions from its design. For such a system, C++ or any successor would seem too large as all the languages at the time demanded a larger footprint than what was desired. The team thus had to think in a different way to go about solving all these problems.

Co-founder of Sun Microsystems, Bill Joy, envisioned a language combining the power of Mesa and C in a paper named *Further* he wrote for the engineers at Sun. Gathering ideas, Gosling began work on enhancing C++ and named it "C++ ++ --", a pun on the evolutionary structure of the language's name. The ++ and **--** meant, *putting in* and *taking out stuff*. He soon abandoned the name and called it **Oak**[1] after the tree that stood outside his office.

Table 1: Who's who of the Java technology[1]
Has worked for GT (Green Team), FP (FirstPerson) and JP (Java Products Group)

Name	GT	FP	JP	Details
Lisa Friendly		✓	✓	FirstPerson employee and member of the Java Products Group
John Gage				Science Office (Director), Sun Microsystems
James Gosling	✓	✓	✓	Lead engineer and key architect of the Java technology
Bill Joy				Co-founder and VP, Sun Microsystems; Principal designer of the UC Berkeley, version of the UNIX® OS
Jonni Kanerva		✓		Java Products Group employee, author of The Java FAQ1
Tim Lindholm		✓	✓	FirstPerson employee and member Java Products Group
Scott McNealy				Chairman, President, and CEO of Sun Microsystems
Patrick Naughton	✓	✓		Green Team member, FirstPerson co-founder
George Paolini				Corporate Marketing (Director), Sun's Java Software Division
Kim Polese		✓		FirstPerson product marketing
Lisa Poulson				Original director of public relations for Java technology (Burson-Marsteller)
Wayne Rosing		✓		FirstPerson President
Eric Schmidt				Former Sun Microsystems Chief Technology Officer
Mike Sheridan	✓			Green Team member

The demise of an idea, birth of another

By now, the work on Oak had been significant but come the year 1993, people saw the demise of set-top boxes, interactive TV and the PDAs. A failure that completely ushered the inventors' thoughts to be reinvented. Only a miracle could make the project a success now. And such a miracle awaited anticipation.

National Center for Supercomputing Applications (NCSA) had just unveiled its new commercial web browser for the internet the previous year. The focus of the team, now diverted towards where they thought the "next-wave" of computing would be — the internet. The team then divulged into the realms of creating the same embeddable technology to be used in the web browser space calling it an **applet** — *a small application*. Keeping all of this in mind, the team created a list of features tackling the C++ problems. In their opinion, the project should ...

- .. be simple and gather tested fundamentals and features from the earlier languages in it,
- .. have standard sets of APIs with basic and advanced features bundled with the language,

- .. get rid of concepts requiring direct manipulation of hardware (in this case, memory) to make the language safe,
- .. be platform independent and may written for every platform once (giving birth to the WORA idiom),
- .. be able to manipulate network programming out-of-the-box,
- .. be embeddable in web browsers, and ...
- .. have the ability for a single program to multi-task and do multiple things at the same time.

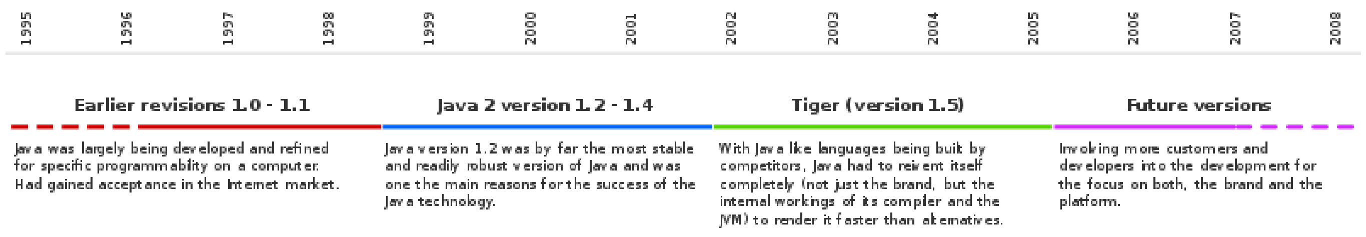
The team now needed a proper identity and they decided on naming the new technology they created Java ushering a new generation of products for the internet boom. A by-product of the project was a cartoon named "Duke" created by Joe Parlang which became its identity then.

Finally at the SunWorld^{TM} conference, Andreesen unveiled the new technology to the masses. Riding along with the explosion of interest and publicity in the Internet, Java quickly received widespread recognition and expectations grew for it to become the dominant software for browser and consumer applications.[2]

Initially Java was owned by Sun Microsystems, but later it was released to open source; the term Java was a trademark of Sun Microsystems. Sun released the source code for its HotSpot Virtual Machine and compiler in November 2006, and most of the source code of the class library in May 2007. Some parts were missing because they were owned by third parties, not by Sun Microsystems. The released parts were published under the terms of the GNU General Public License, a free software license.

Versions

Unlike C and C++, Java's growth is pretty recent. Here, we'd quickly go through the development paths that Java took with age.


1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
Earlier revisions 1.0 - 1.1
Java 2 version 1.2 - 1.4
Tiger (version 1.5)
Future versions
Java was largely being developed and refined for specific programmability on a computer. Had gained acceptance in the Internet market.
Java version 1.2 was by far the most stable and readily robust version of Java and was one the main reasons for the success of the Java technology.
With Java like languages being built by competitors, Java had to reinvent itself completely (not just the brand, but the internal workings of its compiler and the JVM) to render it faster than alternatives.
Invoking more customers and developers into the development for the focus on both, the brand and the platform.

Development of Java over the years. From version 1.0 to version 1.7, Java has displayed a steady growth.

Initial Release (versions 1.0 and 1.1)

Introduced in 1996 for the Solaris, Windows, Mac OS Classic and Linux, Java was initially released as the Java Development Kit 1.0 (JDK 1.0). This included the Java runtime (the virtual machine and the class libraries), and the development tools (e.g., the Java compiler). Later, Sun also provided a runtime-only package, called the Java Runtime Environment (JRE). The first name stuck, however, so usually people refer to a particular version of Java by its JDK version (e.g., JDK 1.0).

Java 2 (version 1.2)

Introduced in 1998 as a quick fix to the former versions, version 1.2 was the start of a new beginning for Java. The JDKs of version 1.2 and later versions are often called *Java 2* as well. For example, the official name of JDK 1.4 is *The Java(TM) 2 Platform, Standard Edition version 1.4*.

Major changes include:

- Rewrite the event handling (add Event Listeners)
- Change Thread synchronizations
- Introduction of the JIT-Just in time compilers

Kestrel (Java 1.3)

Released in 8 May 2000. The most notable changes were:

- HotSpot JVM included (the HotSpot JVM was first released in April, 1999 for the J2SE 1.2 JVM)
- RMI was modified to support optional compatibility with CORBA
- JavaSound
- Java Naming and Directory Interface (JNDI) included in core libraries (previously available as an extension)
- Java Platform Debugger Architecture (JPDA)
- Synthetic proxy classes

Merlin (Java 1.4)

Released in 6 February 2002, Java 1.4 has improved programmer productivity by expanding language features and available APIs:

- Assertion
- Regular Expression
- XML processing
- Cryptography and Secure Socket Layer (SSL)
- Non-blocking I/O (NIO)
- Logging

Tiger (version 1.5.0; Java SE 5)

Released in September 2004

Major changes include:

- Generics - Provides compile-time type safety for collections :and eliminates the drudgery of casting.
- Autoboxing/unboxing - Eliminates the drudgery of manual conversion between primitive types (such as int) and wrapper types (such as Integer).
- Enhanced for - Shorten the for loop with Collections use.
- Static imports - Lets you import all the static part of a class.
- Annotation/Metadata - Enabling tools to generate code and deployment descriptors from annotations in the source code. This leads to a "declarative" programming style where the programmer says what should be done and tools emit the code to do it. Annotations can be inspected through source parsing or by using the additional reflection APIs added in Java 5.
- JVM Improvements - Most of the run time library is now mapped into memory as a memory image, as opposed to being loaded from a series of class files. Large portion of the runtime libraries will now be shared among multiple JVM instances.

Mustang (version 1.6.0; Java SE 6)

Released on 11 December 2006.

What's New in Java SE 6:

- Web Services - First-class support for writing XML web service client applications.

- Scripting - You can now mix in JavaScript technology source code, useful for prototyping. Also useful when you have teams with a variety of skill sets. More advanced developers can plug in their own scripting engines and mix their favorite scripting language in with Java code as they see fit.

- Database - No more need to find and configure your own JDBC database when developing a database application. Developers will also get the updated JDBC 4.0, a well-used API with many important improvements, such as special support for XML as an SQL datatype and better integration of Binary Large OBjects (BLOBs) and Character Large OBjects (CLOBs) into the APIs.

- More Desktop APIs - GUI developers get a large number of new tricks to play like the ever popular yet newly incorporated SwingWorker utility to help you with threading in GUI apps, JTable sorting and filtering, and a new facility for quick splash screens to quiet impatient users.

- Monitoring and Management - The really big deal here is that you don't need to do anything special to the startup to be able to attach on demand with any of the monitoring and management tools in the Java SE platform.

- Compiler Access - Really aimed at people who create tools for Java development and for frameworks like JavaServer Pages (JSP) or Personal Home Page construction kit (PHP) engines that need to generate a bunch of classes on demand, the compiler API opens up programmatic access to javac for in-process compilation of dynamically generated Java code. The compiler API is not directly intended for the everyday developer, but for those of you deafened by your screaming inner geek, roll up your sleeves and give it a try. And the rest of us will happily benefit from the tools and the improved Java frameworks that use this.

- Pluggable Annotations allows programmer to write annotation processor so that it can analyse your code semantically before javac compiles. For example, you could write an annotation processor that verifies whether your program obeys naming conventions.

- Desktop Deployment - At long last, Java SE 6 unifies the Java Plug-in technology and Java WebStart engines, which just makes sense. Installation of the Java WebStart application got a much needed makeover.

- Security - Java SE 6 has simplified the job of its security administrators by providing various new ways to access platform-native security services, such as native Public Key Infrastructure (PKI) and cryptographic services on Microsoft Windows for secure authentication and communication, Java Generic Security Services (Java GSS) and Kerberos services for authentication, and access to LDAP servers for authenticating users.

- The -lities: Quality, Compatibility, Stability - Bug fixes ...

Dolphin (version 1.7.0; Java SE 7)

Released on 28 July 2011.

Feature additions for Java 7 include:

- JVM support for dynamic languages, following the prototyping work currently done on the Multi Language Virtual Machine
- Compressed 64-bit pointers Available in Java 6 with -XX:+UseCompressedOops (http://www.oracle.com/technetwork/java/javase/tech/vmoptions-jsp-140102.html)
- Small language changes (grouped under a project named Coin):
  - Strings in switch
  - Automatic resource management in try-statement
  - Improved type inference for generic instance creation
  - Simplified varargs method declaration
  - Binary integer literals
  - Allowing underscores in numeric literals
  - Catching multiple exception types and rethrowing exceptions with improved type checking
- Concurrency utilities under JSR 166
- New file I/O library to enhance platform independence and add support for metadata and symbolic links. The new packages are java.nio.file and java.nio.file.attribute
- Library-level support for Elliptic curve cryptography algorithms
- An XRender pipeline for Java 2D, which improves handling of features specific to modern GPUs
- New platform APIs for the graphics features originally planned for release in Java version 6u10
- Enhanced library-level support for new network protocols, including SCTP and Sockets Direct Protocol
- Upstream updates to XML and Unicode

Lambda (Java's implementation of lambda functions), Jigsaw (Java's implementation of modules), and part of Coin were dropped from Java 7.

Spider (version 1.8.0; Java SE 8)

Java 8 was released on 18 March 2014, and included some features that were planned for Java 7 but later deferred.

Work on features was organized in terms of JDK Enhancement Proposals (JEPs).

- JSR 335, JEP 126: Language-level support for lambda expressions (officially, lambda expressions; unofficially, closures) under Project Lambda which allow the addition of methods to interfaces without breaking existing implementations. There was an ongoing debate in the Java community on whether to add support for lambda expressions. Supporting lambda expressions also allows the performance of functional-style operations on streams of elements, such as MapReduce-inspired transformations on collections. Default methods allow an author of an API to add new methods to an interface without breaking the old code using it. Although it was not their primary intent, default methods also allow multiple inheritance of behavior (but not state).
- JSR 223, JEP 174: Project Nashorn, a JavaScript runtime which allows developers to embed JavaScript code within applications
- JSR 308, JEP 104: Annotation on Java Types
- Unsigned Integer Arithmetic
- JSR 337, JEP 120: Repeating annotations
- JSR 310, JEP 150: Date and Time API
- JEP 178: Statically-linked JNI libraries
- JEP 153: Launch JavaFX applications (direct launching of JavaFX application JARs)
- JEP 122: Remove the permanent generation

References

1. "Java Technology: The Early Years". Sun Microsystems. http://java.sun.com/features/1998/05/birthday.html. Retrieved 9 May 2008.
2. "History of Java". Lindsey, Clark S.. http://www.particle.kth.se/~lindsey/JavaCourse/Book/Part1/Java/Chapter01/history.html. Retrieved 7 May 2008.
3. Structural syntax is a linear way of writing code. A program is interpreted usually at the first line of the program's code until it reaches the end. One cannot hook a later part of the program to an earlier one. The flow follows a linear top-to-bottom approach.