**1. Reading Assignment: A Short History of Java**

** Task: Read about the history and development of Java.**

** Link:** [**http://sunsite.uakom.sk/sunworldonline/swol-07-1995/swol-07-java.html**](http://sunsite.uakom.sk/sunworldonline/swol-07-1995/swol-07-java.html)

|  |  |
| --- | --- |
| **Java: The inside story**  **We interview Java's creators to find what they had in mind**  *By Michael O'Connell* | [SunWorld](http://sunsite.uakom.sk/sunworldonline/swol-07-1995/index.html) **July  1995**  [Next story] [[Table of Contents]](http://sunsite.uakom.sk/sunworldonline/swol-07-1995/index.html) [[Search]](http://sunsite.uakom.sk/search.html/) [Subscribe to SunWorld, it's free!](http://sunsite.uakom.sk/sunworldonline/common/swol-subscribe.html) |

**Abstract**

Poised to fill World Wide Web browsers everywhere with animation, audio, and real-time interactivity, Sun's Java language has survived an odyssey through consumer electronics, PDAs, set-top boxes, and CD-ROMs. While some of these areas may yet be exploited by the language formerly known as Oak, the Internet is Java's launch pad. How'd it get there, and what's its destiny? Will it successfully cross over into the (gasp!) non-Unix marketplace?

(How bad is your Java habit? Check out our [Java survey](http://sunsite.uakom.sk/sunworldonline/swol-07-1995/swol-07-javasurvey.html) and tell us what *you* are doing with Java.)

*Note: This article was published in July 1995*

|  |
| --- |
| [Mail this article to a friend](http://sunsite.uakom.sk/sunworldonline/cgi-bin/swol-mailfriend.cgi?/swol-07-1995/swol-07-java.html+Java:_The_inside_story+Michael_O+feat) |

One would think that a key component in any business's current strategy for success would have been deliberately created based upon a clearly defined mission. But in the case of Sun Microsystems Inc.'s [Java](http://java.sun.com/) programming language -- the environment that turns static Web pages into interactive, dynamic, animated documents bolstered by distributed, platform-independent applications -- it seems the solution preceded the problem.

As Java creator James Gosling explained in a recent interview with *SunWorld Online*, the genesis of Sun's Web-enhancing technology can be traced to early 1991, when a small group of Sun engineers formed to explore opportunities in the [consumer electronics](http://sunsite.uakom.sk/sunworldonline/swol-07-1995/swol-07-java.html" \l "Lessons) market. At the time, the World Wide Web was still in the drawing rooms.

**Consumers vs. engineers**  
"We were trying to build a distributed system that would make sense as a business [product] ... to sell modern software technology to consumer electronics" manufacturers, Gosling says.

Gosling, 40, joined Sun in 1984 (coming from IBM's research division) and soon afterward began work on the technically impressive but commercially unsuccessful NeWS windowing system. He also wrote GOSMACS, the first EMACS text editor implementation in C.

During this consumer electronics effort, eventually referred to as the "Green" project, Gosling and fellow project engineers learned a great deal about the value of qualities such as reliability, cost, standards, and simplicity -- top priorities in the consumer marketplace. (See the [timeline sidebar](http://sunsite.uakom.sk/sunworldonline/swol-07-1995/swol-07-java.html" \l "Evolution) for additional details.) In contrast to workstation users, who typically want lots of power and will tolerate (and sometimes seemingly demand) high prices, steep learning curves and various bugs in exchange, consumers demand low-cost, bug-free and relatively simple, easy-to-use products.

"Consumers don't care which CPU is inside," says Gosling. They don't appreciate big or powerful RISC-based processors, which are "expensive and proprietary." To compete in the consumer electronics market, companies "treat CPUs as commodities" that can be swapped for lower-cost alternatives nearly instantaneously, and invest resources into backward compatibility and adherence to established standards in long- lived machines, such as toasters and televisions.

Gosling notes that just as modern toasters with embedded electronics employ the same basic user interface as his mother's 42 year-old toaster (which "still works just fine"), so must other consumer electronics products. Television followed a similar path of backward- compatibility when color broadcasts began (today's TV signals can be viewed on 1950s-era black-and-white sets), and faces its next challenge in making the move from analog to digital signals.

Advertisements

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

**Starting with C++**  
To make development a more platform-neutral process (and thus accommodate the consumer market's demand for CPU flexibility), Gosling began by extending the C++ compiler. Eventually, however, he realized that even with lots of extras, C++ would not suffice. Thus, Oak was conceived in mid-1991. (The name came to Gosling when, while creating a directory for the new language, he glanced out his window, and spotted a tree. But the name didn't survive a trademark search, and was dropped in favor of Java.)

"All along, the language was a tool, not the end," Gosling says. "This was nice in a number of ways. The goal was never 'Let's take on C++,' [but] to build a system that would let us do a large, distributed, heterogeneous network of consumer electronic devices all talking to each other."

In the fall of 1992, after what then-project engineer Patrick Naughton characterizes as "massive amounts of hacking on Oak, the Green OS, the UI, [and the] hardware," among other things, the Green project delivered "\*7" (as in the star on a telephone keypad), the PDA-like device that Gosling calls a "handheld remote control."

"In 18 months, we did the equivalent of what 75-people organizations at Sun took three years to do," boasts Naughton -- "an operating system, a language, a toolkit, an interface, a new hardware platform, three custom chips...using new risky technology at every turn. We pulled out all our teeth and put them in each others' mouths."

Naughton, 30, was the project lead on Sun's OpenWindows user environment before joining the secret Green team.

The \*7's small form factor helped emphasize the small size and efficiency of the code, which was the core technology. The product was demonstrated around Sun and impressed important people like Scott McNealy and Bill Joy, but the next step was uncertain.

**Request for proposal ("Instead of writing a few**   
papers and moving on with life...")

While the team was working on Oak and \*7, team members Ed Frank (hardware/technology) and Mike Sheridan (business planning) wrote business and technology road maps for a company in the mold of Dolby Labs that would create and license technology and get its logo alongside Dolby's on consumer electronics products. They had finished several versions of the plans by the time of the \*7 demo. But in early 1993, as Sun weighed Java's options, the Green team (now incorporated as FirstPerson Inc.) got wind of a request for proposal from Time-Warner for a set-top box operating system and video-on-demand technology. "A perfect fit," recalls Gosling.

FirstPerson quickly zeroed in on the set-top box OS market, and placed a bid with Time-Warner. But despite having been told that they had the best technology, Sun did not win the bid, due to what Gosling and Naughton characterize as wholly non-technical reasons, such as business politics. "[SGI's Jim] Clark sold his sword to get the deal," Naughton says.

FirstPerson kept trying to pursue set-top boxes until early 1994, when it concluded that "the market wasn't real," Gosling says. "A lot of people hyped things beyond reason." Apparently, the interactive TV market still isn't ripe. Two recent examples: An interactive cable TV trial of 50,000 homes in Omaha, NE put together by U.S. West (network), 3DO (set-top boxes), and DEC (video servers) was cancelled this spring after two years; and Viacom just stopped work on its full-scale test in the San Francisco bay area -- to refocus on applying its miles of cable TV lines to a tried and true market: telephone service.

Naughton says he waged an eventually successful campaign to stop pursuing set-top boxes and instead focus on online services, CD-ROMs, and desktop platforms. FirstPerson dissolved, and about half of its staff moved to Sun Interactive to develop digital video data servers. But a few people still pursued applying Java's technology to network-based desktop systems.

**Spinning into the Web**  
By mid-1994, the World Wide Web was big. "We realized we could build a really cool browser," says Gosling. "It was one of the few things in the client/server mainstream that needed some of the weird things we'd done: architecture-neutral, real-time, reliable, secure -- issues that weren't terribly important in the workstation world. So we built a browser."

By early fall, Naughton and fellow Sun engineer Jonathan Payne finished writing WebRunner, a Web browser written using the Java language. This early incarnation of HotJava showed off Java in a new light, and a demo impressed SunLabs director Bert Sutherland and Eric Schmidt, Sun chief technology officer -- no doubt in part because they could envision Sun reaping rewards.

**What it all means**  
"The browser equals something that creates a market" for tools, servers, development environments, Gosling says. And Java plays a key role in those tools. "In the pre-Java world, you look at the WWW world, and a page is essentially a piece of paper. In the Java world, a browser becomes a framework. Content providers are empowered to describe behavior and data formats and everything."

More generically, Gosling envisions Java will make people rethink what client/server computing is about. "In the standard model, you have some databases, write a bunch of clients that interact with the database, and build some front end." In this model it can be difficult to organize multiple systems and maintain upgrades, especially when they come from different places, Gosling says.

With Java and Web tools, in contrast, you have inherent organization, Gosling says. "If you build the client side of an application in Java, then launching a client app becomes just switching to a page. Installing is trivial -- just put it on a Web server. And there are no ports, just one version of the application." Already, Gosling says, lots of companies organize databases as Web pages using the Common Gateway Interface (CGI) -- the specified standard for running external programs under an HTTP server.

**Looking ahead**  
At this point, Gosling says the Java language is fairly solid, and he doesn't see any major changes, just some fleshing out. It's the browser that needs work, and that's where the team's efforts are largely concentrated. Gosling says it should be complete by the end of summer. What then? "We've been building a tool, now we'd like to use it," says Gosling -- to build some commercial products, including Web authoring tools.

And none too soon. After the browser, "Sun's biggest challenge is authoring tools" to help develop content, says Dwain Aidala, VP and general manager of Mitsubishi Electronics' North American Multimedia Business Center.

Aidala, whose company has been working with the Java technology in embedded systems for the last two years, says Sun also should make Java a bit lighter and expand its potential applications beyond the Internet and the Web. "Java is limited primarily by how small they can make the interpreter," Aidala says. "It has the potential to go anywhere there's networked processors. ... Telco, CATV, closed systems ... all in the future."

Indeed, Sun acknowledges its efforts to employ Java technology in interactive televisions/set-top boxes, handheld devices, PDAs, telephones, VCRs -- even light switches. "The Internet is the first platform, the perfect way" to introduce Java, says Kim Polese, senior product manager of Java and HotJava.

Still, Java has a ways to go. Of the 713 people who filled out *SunWorld Online* surveys at its June prototype Web site, only 13 people (1.8 percent) use HotJava -- currently the only choice for taking advantage of Java [applets](http://java.sun.com/applets/) -- as their primary browser. Netscape Communications' agreement to license Java technology means a Java- aware version of the Netscape Navigator browser could appear around the end of the year; with the number-one browser up to speed, Java will become a common component in the broader Web community.

**Competition? What competition?**  
Sun insists it has no direct competition. Although nobody is advertising an equal technology, languages such as [Kaleida Labs' ScriptX](http://www.kaleida.com/) and [Lingo](http://www.macromedia.com/Tools/DMS/index.html) (Macromedia Director's animation scripting language) have come up in the same breath as Java. And General Magic offers somewhat similar technology, its [Telescript](http://www.genmagic.com/MagicCapDocs/Concepts/Telescript.html) messaging and agent-based system. But as Polese explains, Telescript differs greatly from Java.

Instead of the server sending information based on client requests, the servers sends things when the *server* deems appropriate, so code comes to clients without clients asking for it. This raises security questions, Polese says. Since Telescript is aimed at phone networks and PDAs, not desktops, the security issues may not apply.

When one considers the applications of the technology, ever-present Microsoft enters the ring. Beyond the typical browser companies or service providers, and beyond Microsoft's monolithic force and looming Microsoft Network, Microsoft has Visual Basic. While this language lacks the portability of Java, it can run on any Windows-based PC. And while VB applications may never be as light as the heaviest Java applet, they are downloadable. "VBA [Visual Basic for Applications, a built-in programming language for MS applications that replaces older macro languages and provides a unified programming interface to the outside world] is interpreted, extensible and downloadable," says HotJava co-author Patrick Naughton.

"Visual Basic can and will do what Java does," says Naughton, who is now the vice president of technology at [Starwave Corp.](http://www.starwave.com/) (Starwave is Microsoft co-founder Paul Allen's Seattle-based interactive consumer products and services company and maker of the popular ESPNET SportsZone Web site, among others.) "Visual Basic already has 30 million users." Who cares about cross-platform issues when you own the PC market?

"Visual Basic makes up for its inferiority as a programming language for serious object oriented design by having a easy to use, visual application construction tool which is already in use by a massive developer community."

"The only thing it doesn't have that matters is security. That's its Achilles' heel -- it's gonna kill 'em." Still, Naughton adds, "PC viruses have been around for 15 years" and haven't prevented the dominance of Windows and DOS. "The issue is that today's viruses have been spread by the rather slow method of floppy disks and downloads from BBSes. Once MSN [the Microsoft Network, due to launch August 24] has people downloading VBA chunks of code as a matter of course, the virus threat is more tangible." In an environment that downloads things for a living, security takes on paramount importance.

In addition to Visual Basic, Microsoft hopes to deliver a tight integration of online network-based services with CD-ROMs. This would allow things like Encarta CD-based encyclopedias to be kept current via supplements seamlessly added via the online network.

"I think this is what Microsoft Network is going to be," says Mark Winther, VP of worldwide telecommunication at IDC's Link Resources Corp., a New York City-based market research and analysis company focusing on the consumer information market. "It doesn't give you live, dynamic, 3-D home pages, but is very powerful and close to what the mass market wants."

Despite all its strengths, Microsoft's concern about Java indicates Sun has the upper hand at this point. "Bill [Gates] knows about and asks about it," says Naughton. "Microsoft sent two guys to pick my brain."

**Marketing strategy**  
Rather than hoarding the technology, Sun has realized the importance of generating broad product interest and acceptance, and therefore freely offers the binaries -- and even the source code -- of key Java components via the Internet.

Sun plans to license Java technology widely to companies such as Netscape that offer Web browsers, online service providers, and software OEMs to "make the long term more solid," Polese says.

"The basic strategy is to license Java to people who have a need for network-centric applications," says Eric Schmidt, Sun's chief technology officer. "The first and obvious target is the browser world. Nothing in the design of Java limits it to Unix or any other operating system. ... It needs to be on all [major] platforms to be successful, and we are going to make sure that happens."

Secondly, Sun is working with third parties to build development tools and object libraries, Polese says, noting that some of these tools are designed for nonprogrammers and offer "completely WYSIWYG" interfaces that let Web creators do things such as drag and drop images and other objects into pages.

"There's no reason why layers can't be built on top" of Java, ranging from scripting tools to more sophisticated tools such as [RAD Technologies' PowerMedia](http://www.batnet.com/RAD/) authoring tool or [Dimension X's](http://www.dnx.com/) animation tools, Polese says.

"What we would really like to see are new types of applications developed using this technology," Schmidt says. "We are trying to avoid the fate of NeWS [Sun's proprietary windowing environment that lost a standards battle with the X Window System] by working more aggressively with everyone in the industry. I think the terms are low enough and the value high enough that we have a good chance of having most of the movers and shakers sign up."

Perhaps these combined efforts will cause tomorrow's Internet -- as well as other networked computer environments -- to overflow with ubiquitous Java objects.

"The people [Sun] needs to market Java to are not those reading [[programming-related newsgroups]](http://java.sun.com/mail.html)," says Naughton, but less sophisticated users. "If average consumer can see these things and use them, they'll demand it. Naughton says Sun needs to land deals with major commercial players such as America Online and CompuServe.

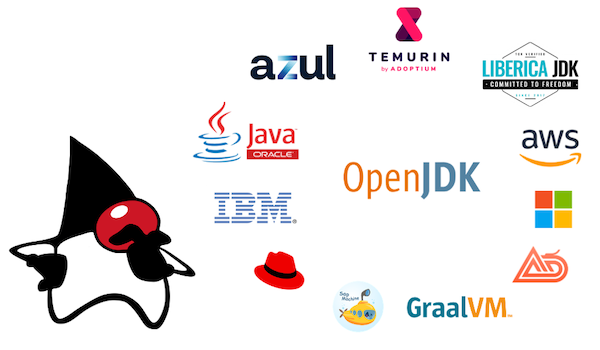
"The key is not Sun, but how many service providers and publishers use Java on their servers," says IDC/Link Resources' Winther. "I see no reason why an increasing number of new sites won't employ it. ... It's real powerful."

**3. Reading Assignment: Which Version of JDK Should I Use?**

** Task: Find out which JDK version is right for you.**

** Link: https://whichjdk.com/**

|  |  |
| --- | --- |
|  |  |

****

**To build and run Java applications, a Java Compiler, Java Runtime Libraries, and a Virtual Machine are required that implement the Java Platform, Standard Edition (“Java SE”) specification.**

**The** [**OpenJDK**](https://openjdk.java.net) **is the open source reference implementation of the Java SE Specification, but it is only the source code. Binary distributions are provided by different vendors for a number of supported platforms. These distributions differ in licenses, commercial support, supported platforms, and update frequency.**

**This site gives independent, yet opinionated recommendations.**

**TL;DR**

**✅ Recommendation: Use** [**Adoptium Eclipse Temurin 21**](https://whichjdk.com/#adoptium-eclipse-temurin) **and ensure that your local version matches the CI and production version.**

**Releases**

**Under the current** [**JDK release model**](https://openjdk.java.net/projects/jdk/)**, a new feature release with a new major version number is planned every six months, in March and September. Additionally, there are quarterly bug fix updates.**

**Every two years, the September release will be a Long-Term-Support (LTS) release, which gets updates for at least three years.**

| **JDK Version** | **Type** | **Release Date** | **Highlights** | **Recommendation** |
| --- | --- | --- | --- | --- |
| [**8**](https://openjdk.java.net/projects/jdk8/) | **LTS** | **03/2014** | **Lambdas** | **Last LTS version under previous release model. Free updates by Oracle** [**ended**](https://www.oracle.com/java/technologies/java-se-support-roadmap.html)**, but still maintained by others. Upgrade to 17 or 21 now!** |
| [**9**](https://openjdk.java.net/projects/jdk9/) | **Feature** | **09/2017** | **Modules** | **New release model was introduced. EOL. Upgrade to 17 or 21 now!** |
| [**10**](https://openjdk.java.net/projects/jdk/10/) | **Feature** | **03/2018** | **var** | **EOL. Upgrade to 17 or 21 now!** |
| [**11**](https://openjdk.java.net/projects/jdk/11/) | **LTS** | **09/2018** | **New HTTP Client** | **Upgrade to 21 now!** |
| [**12**](https://openjdk.java.net/projects/jdk/12/) | **Feature** | **03/2019** |  | **EOL. Upgrade to 21 now!** |
| [**13**](https://openjdk.java.net/projects/jdk/13/) | **Feature** | **09/2019** |  | **EOL. Upgrade to 21 now!** |
| [**14**](https://openjdk.java.net/projects/jdk/14/) | **Feature** | **03/2020** | **Switch expressions** | **EOL. Upgrade to 21 now!** |
| [**15**](https://openjdk.java.net/projects/jdk/15/) | **Feature** | **09/2020** | **Text blocks** | **EOL. Upgrade to 21 now!** |
| [**16**](https://openjdk.java.net/projects/jdk/16/) | **Feature** | **03/2021** | **Records** | **EOL. Upgrade to 21 now!** |
| [**17**](https://openjdk.java.net/projects/jdk/17/) | **LTS** | **09/2021** | **Sealed Classes** | **Supported LTS version. Consider upgrading to 21 in the next months.** |
| [**18**](https://openjdk.java.net/projects/jdk/18/) | **Feature** | **03/2022** | [**UTF-8 by Default**](https://openjdk.java.net/jeps/400) | **EOL. Upgrade to 21 now!** |
| [**19**](https://openjdk.java.net/projects/jdk/19/) | **Feature** | **09/2022** |  | **EOL. Upgrade to 21 now!** |
| [**20**](https://openjdk.java.net/projects/jdk/20/) | **Feature** | **03/2023** |  | **EOL. Upgrade to 21 now!** |
| [**21**](https://openjdk.java.net/projects/jdk/21/) | **LTS** | **09/2023** | [**Pattern Matching**](https://wscp.dev/posts/tech/java-pattern-matching/)**, Virtual Threads** | **Current LTS version.** |
| [**22**](https://openjdk.java.net/projects/jdk/22/) | **Feature** | **03/2024** | [**\_**](https://openjdk.org/jeps/456) | **Stick with 21.** |

**You have to decide if you want to stick with the latest LTS version, or if you go with the latest feature release and upgrade every six months. Both options are okay, but if you’re uncertain, stick with the latest LTS version.**

**The OpenJDK project itself is managed on** [**openjdk.java.net**](https://openjdk.java.net) **where you can find specifications, source code, and mailing lists, but there are no builds that you can download. You need to choose a distribution.**

**Distributions**

* [**OpenJDK builds by Oracle (jdk.java.net)**](https://whichjdk.com/#openjdk-builds-by-oracle-jdkjavanet)
* [**Oracle Java SE Development Kit (JDK)**](https://whichjdk.com/#oracle-java-se-development-kit-jdk)
* [**Adoptium Eclipse Temurin**](https://whichjdk.com/#adoptium-eclipse-temurin)
* [**AdoptOpenJDK**](https://whichjdk.com/#adoptopenjdk)
* [**Azul Zulu**](https://whichjdk.com/#azul-zulu)
* [**Azul Zing**](https://whichjdk.com/#azul-zing)
* [**BellSoft Liberica JDK**](https://whichjdk.com/#bellsoft-liberica-jdk)
* [**IBM Semeru Runtime**](https://whichjdk.com/#ibm-semeru-runtime)
* [**Amazon Corretto**](https://whichjdk.com/#amazon-corretto)
* [**Microsoft Build of OpenJDK**](https://whichjdk.com/#microsoft-build-of-openjdk)
* [**Alibaba Dragonwell**](https://whichjdk.com/#alibaba-dragonwell)
* [**SapMachine**](https://whichjdk.com/#sapmachine)
* [**Red Hat OpenJDK**](https://whichjdk.com/#red-hat-openjdk)
* [**GraalVM**](https://whichjdk.com/#graalvm)

**OpenJDK builds by Oracle (jdk.java.net)**

[**Website**](https://jdk.java.net) **|** [**Releases**](https://jdk.java.net) **| Docker Images (n/a)**

**Oracle provides OpenJDK builds for Linux, macOS and windows in a compressed archive format.**

**These builds will only be updated for a 6-month period. Updates and security patches will not be available after this short period. This also applies for LTS versions! e.g., the latest OpenJDK 11 build was** [**11.0.2+9**](https://jdk.java.net/archive/) **while the current OpenJDK version is** [**11.0.12+7**](https://wiki.openjdk.java.net/display/JDKUpdates/JDK11u)**.**

**⛔️ Recommendation: Do not use *OpenJDK builds by Oracle*, particularly if you plan to stick with LTS versions.**

**Oracle Java SE Development Kit (JDK)**

[**Website**](https://www.oracle.com/java/) **|** [**Releases**](https://www.oracle.com/java/technologies/downloads/archive/) **| Docker Images (n/a)**

**Oracle provides a commercial version of the OpenJDK, which are based on the exactly same sources of the OpenJDK: The Oracle Java SE Development Kit (JDK). Oracle provides updates regular updates and security patches for these builds.**

**The main issue with these builds is Oracle’s licensing policy:**

**Until version 10, builds were published under the** [**Oracle Binary Code License Agreement**](https://www.oracle.com/de/downloads/licenses/binary-code-license.html)**, which effectively allowed the builds to be used for commercial projects.**

**With version 11 to version 16 builds were published under the** [**Oracle Technology Network License Agreement for Oracle Java SE**](https://www.oracle.com/downloads/licenses/javase-license1.html)**, which require a fee-based license for usage in production. This is why many new distributions of the OpenJDK have emerged.**

**Version 17 is published under the** [**Oracle No-Fee Terms and Conditions (NFTC)**](https://www.oracle.com/downloads/licenses/no-fee-license.html)**, which allows the usage of the builds for running *internal business operations*. Unfortunately, the phrase “internal business operations,” is not defined and is a very vague phrase (is a public-facing website running internal business operations?).**

**Also, based on this volatile licensing history, it is not predictable, how future version will be licensed.**

**⛔️ Recommendation: Do not use *Oracle Java SE Development Kit (JDK)* before consulting your lawyer.**

**Adoptium Eclipse Temurin**

[**Website**](https://adoptium.net) **|** [**Releases**](https://adoptium.net/archive.html) **|** [**Docker Images**](https://hub.docker.com/_/eclipse-temurin/)

**Eclipse Adoptium is a top-level project under the Eclipse Foundation, which provides resources and a professional governance model for open source software. The Adoptium Working Group consists of major companies and organizations that have a strategic interest in the Java technology, including Red Hat, IBM, Microsoft, Azul, and the iJUG. The former AdoptOpenJDK project has moved to Eclipse Adoptium.**

**The Adoptium OpenJDK builds are called *Eclipse Temurin* to distinguish the project from the builds.**

**Eclipse Temurin builds are high-quality, vendor-neutral, and TCK-tested under a permissive license.**

**Adoptium states, it will continue to build binaries for LTS releases as long as the corresponding upstream source is actively maintained.**

**✅ Recommendation: *Adoptium Eclipse Temurin* OpenJDK builds are highly recommended.**

**AdoptOpenJDK**

[**Website**](https://adoptopenjdk.net) **|** [**Releases**](https://adoptopenjdk.net/archive.html?variant=openjdk11&jvmVariant=hotspot) **|** [**Docker Images**](https://hub.docker.com/_/adoptopenjdk)

**The AdoptOpenJDK project was the predecessor of Eclipse Adoptium and provided high-quality OpenJDK builds, both for the default HotSpot and the OpenJ9 virtual machine.**

**The website and older releases are kept online to access archived releases.**

**⛔️ Recommendation: Do not use *AdoptOpenJDK* anymore. Use *Adoptium Eclipse Temurin* instead.**

**Azul Zulu**

[**Website**](https://www.azul.com) **|** [**Releases**](https://www.azul.com/downloads/?package=jdk#download-openjdk) **|** [**Docker Images**](https://hub.docker.com/r/azul/zulu-openjdk)

**Azul Zulu Builds of OpenJDK are no-cost, production-ready open-source, TCK-tested, and certified OpenJDK distributions. They are available for a wide range of hardware platforms and operating systems and are compatible with special requirements, such as stripped-down JREs and builds, including OpenJFX and Coordinated Restore at Checkpoint (CRaC).**

**They are supported as part of Azul Platform Core, which provides stabilized security updates for rapid, assured deployment into production and solution-oriented engineering assistance.**

**A downside of these builds is the dependency to a single company, that may suddenly change its license or update policies.**

**✅ Recommendation: *Azul Zulu Builds of OpenJDK* are a good choice.**

**Azul Zing**

[**Website**](https://www.azul.com) **|** [**Releases**](https://www.azul.com/products/prime-roadmap/) **|** [**Docker Images**](https://hub.docker.com/u/azul)

**Azul Zing Builds of OpenJDK (Zing) are commercial optimized builds of OpenJDK, currently marketed as Azul Platform Prime. Zing is free for evaluation but requires a commercial contract with Azul Systems for production use.**

**Zing takes OpenJDK as its base and replaces several key components with optimized versions. The major additions are the C4 Pauseless Garbage Collector (the only generational, production tested pauseless garbage collection available for all major Java versions, including Java 8 and 11), the Falcon JIT Compiler (optimizes code for faster throughput, lower response latencies, and greater carrying capacity), the ReadyNow Warmup Optimizer (learns from previous runs of your application to bring applications to full speed as quickly as possible), and Azul Optimizer Hub (a separate component that offloads JIT compilation from your client machines and lets JVMs learn from each other to reach maximum speed as quickly as possible).**

**Zing is a good choice for latency-sensitive applications that need to guarantee low median latency and minimum latency outliers, applications that aggressively scale up and down and need to be ready to handle traffic as soon as possible, and large fleets of JVMs running an application where the cost of infrastructure is an issue.**

**⚠️ Recommendation: Consider *Azul Zing / Azul Platform Prime* when GC pause times, slow warmup, and large on-prem infrastructure or Cloud costs are a problem. Do not use it in production without a license.**

**BellSoft Liberica JDK**

[**Website**](https://bell-sw.com) **|** [**Releases**](https://bell-sw.com/pages/downloads/?) **|** [**Docker Images**](https://hub.docker.com/u/bellsoft)

**Similar to Azul, BellSoft has specialized in professional Java technologies and commercial support for JDK. Also, BellSoft has a high industry reputation and is engaged in various working groups to evolve the Java platform.**

**BellSoft provides open source OpenJDK builds called *Liberica JDK* for pretty much all operating systems and architectures.**

**The popular Spring Boot framework chose Liberica JDK as runtime for their** [**buildpack**](https://github.com/paketo-buildpacks/bellsoft-liberica)**.**

**A downside of these builds is the dependency to a single company, that may suddenly change its license or update policies.**

**✅ Recommendation: *BellSoft Liberica JDK* builds are a good choice.**

**IBM Semeru Runtime**

[**Website**](https://developer.ibm.com/languages/java/semeru-runtimes/) **|** [**Releases**](https://developer.ibm.com/languages/java/semeru-runtimes/downloads/) **| Docker Images (n/a)**

**IBM developed its own version of the Java Virtual Machine, called J9 and it was open-sourced as *Eclipse OpenJ9*. It is an alternative to the default HotSpot Java Virtual Machine, but it has never gained much popularity.**

**IBM now provides builds called *Semeru Runtime* based on the Eclipse OpenJ9 Java Virtual Machine and some OpenJDK class libraries. OpenJ9 has a** [**low memory footprint and starts fast with shared classes**](https://www.eclipse.org/openj9/performance/)**, but lower throughput compared to Hotspot Virtual Machine.**

**⚠️ Recommendation: Use *IBM Semeru Runtime* only if you know that you need the OpenJ9 Virtual Machine.**

**Amazon Corretto**

[**Website**](https://aws.amazon.com/corretto/) **|** [**Releases**](https://aws.amazon.com/corretto/) **|** [**Docker Images**](https://hub.docker.com/_/amazoncorretto)

**Since Oracle changed the support and license policy for its OpenJDK builds, major cloud providers decided to establish their own managed OpenJDK builds and providing long-term updates. Apparently, this is to avoid risks, especially lawsuits against Oracle.**

**In 2018, AWS published *Corretto*, yet another OpenJDK build.**

**AWS includes back ports of bug fixes from newer OpenJDK versions and** [**claims**](https://aws.amazon.com/corretto/faqs/) **that they would add patches that might not yet be integrated in the OpenJDK project. Amazon has implemented an alternative** [**crypto provider**](https://github.com/corretto/amazon-corretto-crypto-provider) **that has been optimized for their services. It is** [**planned**](https://aws.amazon.com/blogs/opensource/introducing-amazon-corretto-crypto-provider-accp/) **to be used as the default crypto implementation in Corretto.**

**Amazon provides releases for major development platforms and an optimized version for its own Amazon Linux 2.**

**✅ Recommendation: *Corretto* builds are a good choice, particularly if you run Java applications directly on Amazon Linux 2 in AWS.**

**Microsoft Build of OpenJDK**

[**Website**](https://www.microsoft.com/openjdk) **|** [**Releases**](https://docs.microsoft.com/en-us/java/openjdk/download) **|** [**Docker Images**](https://docs.microsoft.com/en-us/java/openjdk/containers)

**In 2021, Microsoft published *Microsoft Build of OpenJDK*, yet another OpenJDK build.**

**Microsoft may include back ports of bug fixes from newer OpenJDK versions and claims that they would add patches that might not yet be integrated in the OpenJDK project.**

**Microsoft provides releases for major development platforms.**

**⚠️ Recommendation: Use *Microsoft Build of OpenJDK*, only if you run Java applications directly on Azure. There are more established options available.**

**Alibaba Dragonwell**

[**Website**](http://dragonwell-jdk.io) **|** [**Releases**](http://dragonwell-jdk.io) **|** [**Docker Images**](https://github.com/alibaba/dragonwell11/wiki/Use-Dragonwell-11-docker-images)

**Alibaba provides an OpenJDK build which includes back ports and some *extra features*.**

**⛔️ Recommendation: Do not use *Alibaba Dragonwell*, unless you are forced by your government.**

**SapMachine**

[**Website**](https://sap.github.io/SapMachine/) **|** [**Releases**](https://github.com/SAP/SapMachine/releases) **|** [**Docker Images**](https://hub.docker.com/_/sapmachine)

**SapMachine is yet another OpenJDK Build, maintained by SAP.**

**⚠️ Recommendation: Use *SapMachine* only if you are running Java applications on SAP servers. There are more established options available.**

**Red Hat OpenJDK**

[**Website**](https://developers.redhat.com/products/openjdk/overview) **|** [**Releases**](https://developers.redhat.com/products/openjdk/download) **|** [**Docker Images**](https://catalog.redhat.com/software/containers/ubi8/openjdk-11/5dd6a4b45a13461646f677f4)

**Red Hat provides OpenJDK builds for LTS versions.**

**⚠️ Recommendation: Use *Red Hat OpenJDK* only if you are running Java applications directly on Red Hat Enterprise Linux. There are more established options available.**

**ojdkbuild**

[**Website**](https://github.com/ojdkbuild/ojdkbuild) **|** [**Releases**](https://github.com/ojdkbuild/ojdkbuild/releases) **| Docker Images (n/a)**

**The project is discontinued. The ojdkbuild project had the goal of providing Windows x86\_64 binaries of OpenJDK that are as close in behaviour to Linux OpenJDK packages as possible, e.g. by using system libraries instead of packaged versions of zlib or OpenSSL. It used the packages included in CentOS. A use case for these builds was to develop Java software on Windows machines and deploy them to Linux servers in production.**

**⛔️ Recommendation: Do not use *ojdkbuild*, as the project is discontinued.**

**GraalVM**

[**Website**](https://www.graalvm.org) **|** [**Releases**](https://github.com/graalvm/graalvm-ce-builds/releases) **|** [**Docker Images**](https://github.com/graalvm/container/pkgs/container/graalvm-ce)

**GraalVM is a fully compliant JDK, but much different from all the others builds.**

**GraalVM was developed by Oracle. It is based on the OpenJDK but includes a new high-performance compiler and a new polyglot virtual machine (can execute code written in different programming languages). It is also possible to create platform-specific native executable that are highly optimized and start extremely fast.**

**🤷 Please** [**share**](https://github.com/whichjdk/whichjdk.com/issues/6) **your experiences with GraalVM in production, so that we can elaborate a validated recommendation.**

**Special Cases**

**Apple Silicon**

**The official support for *macOS/AArch64* was implemented with** [**JEP 391**](https://openjdk.java.net/jeps/391) **in the OpenJDK 17 release.**

**macOS *x64* builds run stable with Rosetta 2, but there is a significant performance drop due to emulation. People that develop on an *Apple Silicon* Mac (like me) should install a native macOS *AArch64* (aka *ARM 64*) build of the JDK.**

**Most distributions have *macOS/AArch64* builds for Java 17+, only.** [**BellSoft Liberica**](https://bell-sw.com/announcements/2021/03/12/Liberica-on-Apple-Silicon/)**, Amazon Corretto, and** [**Azul Zulu**](https://www.azul.com/newsroom/azul-announces-support-of-java-builds-of-openjdk-for-apple-silicon/) **also provide free *macOS/AArch64* builds for Java 8 and Java 11.**

**FAQs**

**What is the best way to install a JDK for local development?**

**Use** [**SDKMAN!**](https://sdkman.io/install)

**To list available JDKs, type**

**sdk list java**

**and install a specific version:**

**sdk install java 21.0.3-tem**

**Validate by checking the version:**

**java --version**

**Which version of Java do I currently have installed?**

**which java**

**`which java` --version**

**On Linux, you might also try**

**sudo update-java-alternatives**

**What is the difference between JDK and JRE?**

**Some distributions provide a JDK (Java Development Kit) and a JRE (Java Runtime Environment) build. A JDK includes everything to *compile, package and run* Java applications, while a JRE only includes the binaries and libraries to *run* Java applications. The JRE is a stripped down version of the JDK, and is smaller in terms of megabytes.**

**If size matters for you, consider creating your own stripped-down runtime using** [**jlink**](https://blog.adoptium.net/2021/10/jlink-to-produce-own-runtime/)**.**

**For local development, you need a JDK. In production you only need a runtime environment, but it is quite common to use the JDK, too.**

**What about Java EE?**

***Java EE (Java Platform, Enterprise Edition)* was renamed to *Jakarta EE*. It is a specification to build server app and frontends. In terms of scope, Jakarta EE can be compared with more modern frameworks like** [**Spring Boot**](https://spring.io/projects/spring-boot)**,** [**Micronaut**](https://micronaut.io)**, and** [**Quarkus**](https://quarkus.io)**, but Jakarta EE feels more complicated.**

**⚠️ Recommendation: Do not start new projects based on *Jakarta EE*. Most people use *Spring Boot*, which is a good choice. Consider *Quarkus*, if you have a strong Java EE background. Consider *Micronaut*, if you like Groovy and Grails.**

**About**

**This site is maintained by** [**Jochen Christ**](https://twitter.com/jochen_christ)**. Any recommendations or opinions represented on this site are personal and based on long-term professional experience. The author is not associated with any of the organizations stated here.**

**Found an error or something is missing? Please** [**raise an issue**](https://github.com/whichjdk/whichjdk.com/issues/new) **or** [**create a pull request**](https://github.com/whichjdk/whichjdk.com/pulls)**.**

**Java and OpenJDK are trademarks or registered trademarks of Oracle and/or its affiliates.**

**4. Reading Assignment: JDK Installation Directory Structure**

** Task: Understand the folder structure and files in the JDK installation.**

** Link: https://docs.oracle.com/javase/8/docs/technotes/tools/windows/jdkfiles.html**

**3 JDK and JRE File Structure**

**This chapter introduces the JDK directories and the files they contain. The file structure of the JRE is identical to the structure of the jre directory in the JDK.**

**This chapter covers the following topics:**

* **[Demos and Samples](https://docs.oracle.com/javase/8/docs/technotes/tools/windows/jdkfiles.html" \l "A1098669)**
* [**Development Files and Directories**](https://docs.oracle.com/javase/8/docs/technotes/tools/windows/jdkfiles.html#A1097730)
* [**Additional Files and Directories**](https://docs.oracle.com/javase/8/docs/technotes/tools/windows/jdkfiles.html#A1097756)

**Demos and Samples**

**Demos and samples that show you how to program for the Java platform are available as a separate download at [Java Downloads](https://www.oracle.com/java/technologies/downloads/).**

**Development Files and Directories**

**This section describes the most important files and directories required to develop applications for the Java platform. Some of the directories that are not required include Java source code and C header files. See [Additional Files and Directories](https://docs.oracle.com/javase/8/docs/technotes/tools/windows/jdkfiles.html" \l "A1097756).**

**jdk-1.8**

**bin**

**java\***

**javac\***

**javap\***

**javah\***

**javadoc\***

**lib**

**tools.jar**

**dt.jar**

**jre**

**bin**

**java\***

**lib**

**applet**

**ext**

**jfxrt.jar**

**localdata.jar**

**fonts**

**security**

**sparc**

**server**

**client**

**rt.jar**

**charsets.jar**

**Assuming the JDK software is installed at /jdk-1.8, here are some of the most important directories:**

**/jdk-1.8**

**Root directory of the JDK software installation. Contains copyright, license, and README files. Also contains src.zip, the archive of source code for the Java platform.**

**/jdk-1.8/bin**

**Executables for all the development tools contained in the JDK. The PATH environment variable should contain an entry for this directory.**

**/jdk-1.8/lib**

**Files used by the development tools. Includes tools.jar, which contains non-core classes for support of the tools and utilities in the JDK. Also includes dt.jar, the DesignTime archive of BeanInfo files that tell interactive development environments (IDEs) how to display the Java components and how to let the developer customize them for an application.**

**/jdk-1.8/jre**

**Root directory of the Java Runtime Environment (JRE) used by the JDK development tools. The runtime environment is an implementation of the Java platform. This is the directory referred to by the java.home system property.**

**/jdk-1.8/jre/bin**

**Executable files for tools and libraries used by the Java platform. The executable files are identical to files in /jdk-1.8/bin. The java launcher tool serves as an application launcher. This directory does not need to be in the PATH environment variable.**

**/jdk-1.8/jre/lib**

**Code libraries, property settings, and resource files used by the JRE. For example rt.jar contains the bootstrap classes, which are the run time classes that comprise the Java platform core API, and charsets.jar contains the character-conversion classes. Aside from the ext subdirectory, there are several additional resource subdirectories not described here.**

**/jdk-1.8/jre/lib/ext**

**Default installation directory for extensions to the Java platform. This is where the JavaHelp JAR file goes when it is installed, for example. This directory includes the jfxrt.jar file, which contains the JavaFX runtime libraries and the localedata.jar file, which contains the locale data for the java.text and java.util packages. See** [**The Extension Mechanism**](https://docs.oracle.com/javase/8/docs/technotes/guides/extensions/index.html)**.**

**/jdk-1.8/jre/lib/security**

**Contains files used for security management. These include the security policy java.policy and security properties java.security files.**

**/jdk-1.8/jre/lib/applet**

**JAR files that contain support classes for applets can be placed in the lib/applet/ directory. This reduces startup time for large applets by allowing applet classes to be preloaded from the local file system by the applet class loader and provides the same protections as though they had been downloaded over the Internet.**

**/jdk-1.8/jre/lib/fonts**

**Font files used by the platform.**

**Additional Files and Directories**

**This section describes the directory structure for Java source code, C header files, and other additional directories and files.**

**jdk-1.8**

**include**

**man**

**src.zip**

**/jdk-1.8/src.zip**

**Archive that contains the source code for the Java platform.**

**/jdk-1.8/include**

**5. Reading Assignment: About Java Technology**

** Task: Read about the basics of Java technology and its components.**

** Link:** [**https://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html**](https://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html)

**About the Java Technology**

**Java technology is both a programming language and a platform.**

**The Java Programming Language**

**The Java programming language is a high-level language that can be characterized by all of the following buzzwords:**

|  |  |
| --- | --- |
| * **Simple** * **Object oriented** * **Distributed** * **Multithreaded** * **Dynamic** | * **Architecture neutral** * **Portable** * **High performance** * **Robust** * **Secure** |

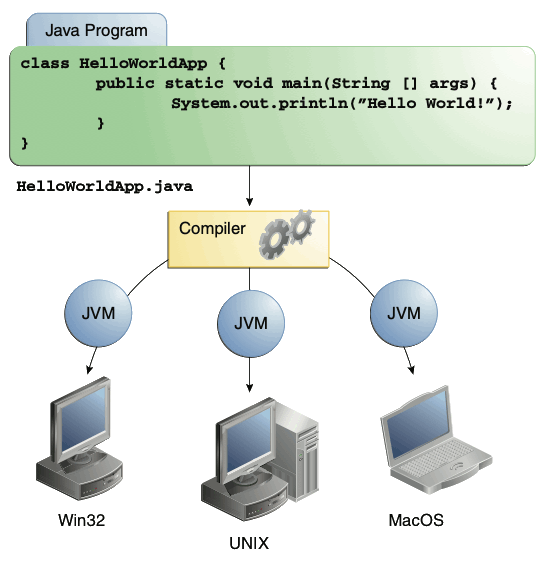
**Each of the preceding buzzwords is explained in** [***The Java Language Environment***](http://www.oracle.com/technetwork/java/langenv-140151.html) **, a white paper written by James Gosling and Henry McGilton.**

**In the Java programming language, all source code is first written in plain text files ending with the .java extension. Those source files are then compiled into .class files by the javac compiler. A .class file does not contain code that is native to your processor; it instead contains *bytecodes* — the machine language of the Java Virtual Machine**[**1**](https://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html#FOOT) **(Java VM). The java launcher tool then runs your application with an instance of the Java Virtual Machine.**

****

**An overview of the software development process.**

**Because the Java VM is available on many different operating systems, the same .class files are capable of running on Microsoft Windows, the Solaris™ Operating System (Solaris OS), Linux, or Mac OS. Some virtual machines, such as the** [**Java SE HotSpot at a Glance**](http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136373.html)**, perform additional steps at runtime to give your application a performance boost. This includes various tasks such as finding performance bottlenecks and recompiling (to native code) frequently used sections of code.**

****

**Through the Java VM, the same application is capable of running on multiple platforms.**

**The Java Platform**

**A *platform* is the hardware or software environment in which a program runs. We've already mentioned some of the most popular platforms like Microsoft Windows, Linux, Solaris OS, and Mac OS. Most platforms can be described as a combination of the operating system and underlying hardware. The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other hardware-based platforms.**

**The Java platform has two components:**

* **The *Java Virtual Machine***
* **The *Java Application Programming Interface* (API)**

**You've already been introduced to the Java Virtual Machine; it's the base for the Java platform and is ported onto various hardware-based platforms.**

**The API is a large collection of ready-made software components that provide many useful capabilities. It is grouped into libraries of related classes and interfaces; these libraries are known as *packages*. The next section,** [**What Can Java Technology Do?**](https://docs.oracle.com/javase/tutorial/getStarted/intro/cando.html) **highlights some of the functionality provided by the API.**

****

**The API and Java Virtual Machine insulate the program from the underlying hardware.**

**As a platform-independent environment, the Java platform can be a bit slower than native code. However, advances in compiler and virtual machine technologies are bringing performance close to that of native code without threatening portability.**

**The terms"Java Virtual Machine" and "JVM" mean a Virtual Machine for the Java platform.**

**7. Reading Assignment: The JVM Architecture Explained**

** Task: Learn about how the Java Virtual Machine (JVM) works.**

** Link:** [**https://dzone.com/articles/jvm-architecture-explained**](https://dzone.com/articles/jvm-architecture-explained)

**The JVM Architecture Explained**

**Click here to check out this overview of the different components of the JVM, along with a very useful diagram!**

**By**

****

[**Jackson Joseraj**](https://dzone.com/users/2863419/jacksondaf.html)

**·**

**Updated Aug. 26, 19 · Tutorial**

**Like** [**(345)**](https://dzone.com/articles/jvm-architecture-explained)

**Save**

[**Tweet**](https://dzone.com/articles/jvm-architecture-explained)

[**Share**](https://www.linkedin.com/sharing/share-offsite/?url=https://dzone.com/articles/jvm-architecture-explained)

**575.0K Views**

**Join the DZone community and get the full member experience.**

[**Join For Free**](https://dzone.com/static/registration.html)

**Every Java developer knows that bytecode will be executed by the JRE (Java Runtime Environment). But many don't know the fact that JRE is the implementation of Java Virtual Machine (JVM), which analyzes the bytecode, interprets the code, and executes it. It is very important, as a developer, that we know the architecture of the JVM, as it enables us to write code more efficiently. In this article, we will learn more deeply about the JVM architecture in Java and different components of the JVM.**

**What Is the JVM?**

**A Virtual Machine is a software implementation of a physical machine. Java was developed with the concept of WORA (*Write Once Run Anywhere*), which runs on a VM. The compiler compiles the Java file into a Java .class file, then that .class file is input into the JVM, which loads and executes the class file.**

[**The Complete Java Master Class Bundle.\***](https://academy.techrepublic.com/sales/the-2020-complete-java-master-class-certification-bundle)

***\*Affiliate link. See***[***Terms of Use***](https://technologyadvice.com/terms-conditions/)**.**

**How Does the JVM Work?**

**As shown in the above architecture diagram, the JVM is divided into three main subsystems:**

1. **ClassLoader Subsystem**
2. **Runtime Data Area**
3. **Execution Engine**

**1. ClassLoader Subsystem**

**Java's** [**dynamic class loading**](http://www.javainterviewpoint.com/use-of-class-forname-in-java/) **functionality is handled by the ClassLoader subsystem. It loads, links. and initializes the class file when it refers to a class for the first time at runtime, not compile time.**

**1.1 Loading**

**Classes will be loaded by this component. BootStrap ClassLoader, Extension ClassLoader, and Application ClassLoader are the three ClassLoaders that will help in achieving it.**

1. **BootStrap** [**ClassLoader**](http://www.javainterviewpoint.com/) **– Responsible for loading classes from the bootstrap classpath, nothing but rt.jar. Highest priority will be given to this loader.**
2. **Extension ClassLoader – Responsible for loading classes which are inside the ext folder (jre\lib).**
3. **Application ClassLoader –Responsible for loading Application Level Classpath, path mentioned Environment Variable, etc.**

**The above ClassLoaders will follow Delegation Hierarchy Algorithm while loading the class files.**

**1.2 Linking**

1. **Verify – Bytecode verifier will verify whether the generated bytecode is proper or not if verification fails we will get the verification error.**
2. **Prepare – For all static variables memory will be allocated and assigned with default values.**
3. **Resolve – All symbolic memory references are replaced with the original references from Method Area.**

**1.3 Initialization**

**This is the final phase of ClassLoading; here, all** [**static variables**](http://www.javainterviewpoint.com/use-of-static-keyword-in-java/) **will be assigned with the original values, and the** [**static block**](http://www.javainterviewpoint.com/java-static-import/) **will be executed.**

**2. Runtime Data Area**

**The Runtime Data Area is divided into five major components:**

1. **Method Area – All the class-level data will be stored here, including static variables. There is only one method area per JVM, and it is a shared resource.**
2. **Heap Area – All the Objects and their corresponding instance variables and arrays will be stored here. There is also one Heap Area per JVM. Since the Method and Heap areas share memory for multiple threads, the data stored is not thread-safe.**
3. **Stack Area– For every thread, a separate runtime stack will be created. For every method call, one entry will be made in the stack memory which is called Stack Frame. All local variables will be created in the stack memory. The stack area is thread-safe since it is not a shared resource. The Stack Frame is divided into three subentities:** 
   1. **Local Variable Array – Related to the method how many local variables are involved and the corresponding values will be stored here.**
   2. **Operand stack – If any intermediate operation is required to perform, operand stack acts as runtime workspace to perform the operation.**
   3. **Frame data – All symbols corresponding to the method is stored here. In the case of any exception, the catch block information will be maintained in the frame data.**
4. **PC Registers – Each thread will have separate PC Registers, to hold the address of current executing instruction once the instruction is executed the PC register will be updated with the next instruction.**
5. **Native Method stacks – Native Method Stack holds native method information. For every thread, a separate native method stack will be created.**

**3. Execution Engine**

**The bytecode, which is assigned to the Runtime Data Area, will be executed by the Execution Engine. The Execution Engine reads the bytecode and executes it piece by piece.**

1. **Interpreter – The interpreter interprets the bytecode faster but executes slowly. The disadvantage of the interpreter is that when one method is called multiple times, every time a new interpretation is required.**
2. **JIT Compiler– The JIT Compiler neutralizes the disadvantage of the interpreter. The Execution Engine will be using the help of the interpreter in converting byte code, but when it finds repeated code it uses the JIT compiler, which compiles the entire bytecode and changes it to native code. This native code will be used directly for repeated method calls, which improve the performance of the system.** 
   1. **Intermediate Code Generator – Produces intermediate code**
   2. **Code Optimizer – Responsible for optimizing the intermediate code generated above**
   3. **Target Code Generator – Responsible for Generating Machine Code or Native Code**
   4. **Profiler – A special component, responsible for finding hotspots, i.e. whether the method is called multiple times or not.**
3. **Garbage Collector: Collects and removes unreferenced objects. Garbage Collection can be triggered by calling System.gc(), but the execution is not guaranteed. Garbage collection of the JVM collects the objects that are created.**

**Java Native Interface (JNI): JNI will be interacting with the Native Method Libraries and provides the Native Libraries required for the Execution Engine.**

**Native Method Libraries: This is a collection of the Native Libraries, which is required for the Execution Engine.**

**8. Reading Assignment: The Java Language Environment: Contents**

** Task: Explore the content and features of the Java language environment.**

** Link: https://www.oracle.com/java/technologies/language-environment.html**

**The Java Language Environment: Contents**

**A White Paper   
  
May 1996   
James Gosling   
Henry McGilton**

1. [**Introduction to Java**](https://www.oracle.com/java/technologies/introduction-to-Java.html#318)
   1. [**Beginnings of the Java Language Project**](https://www.oracle.com/java/technologies/introduction-to-Java.html#943)
   2. [**Design Goals of Java**](https://www.oracle.com/java/technologies/introduction-to-Java.html#334)
      1. [**Simple, Object Oriented, and Familiar**](https://www.oracle.com/java/technologies/introduction-to-Java.html#349)
      2. [**Robust and Secure**](https://www.oracle.com/java/technologies/introduction-to-Java.html#367)
      3. [**Architecture Neutral and Portable**](https://www.oracle.com/java/technologies/introduction-to-Java.html#379)
      4. [**High Performance**](https://www.oracle.com/java/technologies/introduction-to-Java.html#1751)
      5. [**Interpreted, Threaded, and Dynamic**](https://www.oracle.com/java/technologies/introduction-to-Java.html#528)
   3. [**The Java Platform--a New Approach to Distributed Computing**](https://www.oracle.com/java/technologies/introduction-to-Java.html#937)
2. [**Java--Simple and Familiar**](https://www.oracle.com/java/technologies/simple-familiar.html#343) 
   1. [**Main Features of the Java Language**](https://www.oracle.com/java/technologies/simple-familiar.html#1225) 
      1. [**Primitive Data Types**](https://www.oracle.com/java/technologies/simple-familiar.html#376)
      2. [**Arithmetic and Relational Operators**](https://www.oracle.com/java/technologies/simple-familiar.html#406)
      3. [**Arrays**](https://www.oracle.com/java/technologies/simple-familiar.html#410)
      4. [**Strings**](https://www.oracle.com/java/technologies/simple-familiar.html#414)
      5. [**Multi-Level Break**](https://www.oracle.com/java/technologies/simple-familiar.html#429)
      6. [**Memory Management and Garbage Collection**](https://www.oracle.com/java/technologies/simple-familiar.html#2333)
      7. [**The Background Garbage Collector**](https://www.oracle.com/java/technologies/simple-familiar.html#455)
      8. [**Integrated Thread Synchronization**](https://www.oracle.com/java/technologies/simple-familiar.html#457)
   2. [**Features Removed from C and C++**](https://www.oracle.com/java/technologies/simple-familiar.html#4076) 
      1. [**No More Typedefs, Defines, or Preprocessor**](https://www.oracle.com/java/technologies/simple-familiar.html#4078)
      2. [**No More Structures or Unions**](https://www.oracle.com/java/technologies/simple-familiar.html#4083)
      3. [**No Enums**](https://www.oracle.com/java/technologies/simple-familiar.html#5627)
      4. [**No More Functions**](https://www.oracle.com/java/technologies/simple-familiar.html#5642)
      5. [**5No More Multiple Inheritance**](https://www.oracle.com/java/technologies/simple-familiar.html#4090)
      6. [**No More Goto Statements**](https://www.oracle.com/java/technologies/simple-familiar.html#4093)
      7. [**No More Operator Overloading**](https://www.oracle.com/java/technologies/simple-familiar.html#4098)
      8. [**No More Automatic Coercions**](https://www.oracle.com/java/technologies/simple-familiar.html#4100)
      9. [**No More Pointers**](https://www.oracle.com/java/technologies/simple-familiar.html#4107)
   3. [**Summary**](https://www.oracle.com/java/technologies/simple-familiar.html#4130)
3. [**Java is Object Oriented**](https://www.oracle.com/java/technologies/object-oriented.html#343) 
   1. [**Object Technology in Java**](https://www.oracle.com/java/technologies/object-oriented.html#2414)
   2. [**What Are Objects?**](https://www.oracle.com/java/technologies/object-oriented.html#1354)
   3. [**Basics of Objects**](https://www.oracle.com/java/technologies/object-oriented.html#6681) 
      1. [**Classes**](https://www.oracle.com/java/technologies/object-oriented.html#1372)
      2. [**Instantiating an Object from its Class**](https://www.oracle.com/java/technologies/object-oriented.html#3121)
      3. [**Constructors**](https://www.oracle.com/java/technologies/object-oriented.html#4517)
      4. [**Methods and Messaging**](https://www.oracle.com/java/technologies/object-oriented.html#4559)
      5. [**Finalizers**](https://www.oracle.com/java/technologies/object-oriented.html#2653)
      6. [**Subclasses**](https://www.oracle.com/java/technologies/object-oriented.html#2600)
      7. [**Java Language Interfaces**](https://www.oracle.com/java/technologies/object-oriented.html#6185)
      8. [**Access Control**](https://www.oracle.com/java/technologies/object-oriented.html#6711)
      9. [**Packages**](https://www.oracle.com/java/technologies/object-oriented.html#1389)
      10. [**Class Variables and Class Methods**](https://www.oracle.com/java/technologies/object-oriented.html#2996)
      11. [**Abstract Methods**](https://www.oracle.com/java/technologies/object-oriented.html#3057)
   4. [**Summary**](https://www.oracle.com/java/technologies/object-oriented.html#2314)
4. [**Architecture Neutral, Portable, and Robust**](https://www.oracle.com/java/technologies/architecture-neutral-portable-robust.html#319) 
   1. [**Architecture Neutral**](https://www.oracle.com/java/technologies/architecture-neutral-portable-robust.html#397) 
      1. [**Byte Codes**](https://www.oracle.com/java/technologies/architecture-neutral-portable-robust.html#402)
   2. [**Portable**](https://www.oracle.com/java/technologies/architecture-neutral-portable-robust.html#269)
   3. [**Robust**](https://www.oracle.com/java/technologies/architecture-neutral-portable-robust.html#882) 
      1. [**Strict Compile-Time and Run-Time Checking**](https://www.oracle.com/java/technologies/architecture-neutral-portable-robust.html#370)
   4. [**Summary**](https://www.oracle.com/java/technologies/architecture-neutral-portable-robust.html#367)
5. [**Interpreted and Dynamic**](https://www.oracle.com/java/technologies/interpreted-dynamic.html#283) 
   1. [**Dynamic Loading and Binding**](https://www.oracle.com/java/technologies/interpreted-dynamic.html#1167) 
      1. [**The Fragile Superclass Problem**](https://www.oracle.com/java/technologies/interpreted-dynamic.html#1170)
      2. [**Solving the Fragile Superclass Problem**](https://www.oracle.com/java/technologies/interpreted-dynamic.html#1173)
      3. [**Run-Time Representations**](https://www.oracle.com/java/technologies/interpreted-dynamic.html#1183)
   2. [**Summary**](https://www.oracle.com/java/technologies/interpreted-dynamic.html#1918)
6. [**Security in Java**](https://www.oracle.com/java/technologies/security-in-java.html#283) 
   1. [**Memory Allocation and Layout**](https://www.oracle.com/java/technologies/security-in-java.html#2762)
   2. [**Security Checks in the Class Loader**](https://www.oracle.com/java/technologies/security-in-java.html#2801)
   3. [**The Byte Code Verification Process**](https://www.oracle.com/java/technologies/security-in-java.html#1056)
      1. [**The Byte Code Verifier**](https://www.oracle.com/java/technologies/security-in-java.html#1063)
   4. [**Security in the Java Networking Package**](https://www.oracle.com/java/technologies/security-in-java.html#1461)
   5. [**Summary**](https://www.oracle.com/java/technologies/security-in-java.html#2513)
7. [**Multithreading**](https://www.oracle.com/java/technologies/multithreading.html#283) 
   1. [**Threads at the Java Language Level**](https://www.oracle.com/java/technologies/multithreading.html#1217)
   2. [**Integrated Thread Synchronization**](https://www.oracle.com/java/technologies/multithreading.html#1222)
   3. [**Multithreading Support--Conclusion**](https://www.oracle.com/java/technologies/multithreading.html#1229)
8. [**Performance and Comparisons**](https://www.oracle.com/java/technologies/performance-comparisons.html#2376) 
   1. [**Performance**](https://www.oracle.com/java/technologies/performance-comparisons.html#2489)
   2. [**The Java Language Compared**](https://www.oracle.com/java/technologies/performance-comparisons.html#1834)
   3. [**A Major Benefit of Java: Fast and Fearless Prototyping**](https://www.oracle.com/java/technologies/performance-comparisons.html#2451)
   4. [**Summary**](https://www.oracle.com/java/technologies/performance-comparisons.html#1902)
9. [**Java Base System and Libraries**](https://www.oracle.com/java/technologies/java-base-system-libraries.html#942) 
   1. [**Java Language Classes**](https://www.oracle.com/java/technologies/java-base-system-libraries.html#942)
   2. [**Input Output Package**](https://www.oracle.com/java/technologies/java-base-system-libraries.html#1227)
   3. [**Utility Package**](https://www.oracle.com/java/technologies/java-base-system-libraries.html#1310)
   4. [**Abstract Window Toolkit**](https://www.oracle.com/java/technologies/java-base-system-libraries.html#1392)

Java Modifiers.

**1.Access Modifiers** - controls the access level

**2.Non-Access Modifiers** - do not control access level, but provides other functionality

Access Modifiers

For **classes**, you can use either public or *default*:

|  |  |  |  |
| --- | --- | --- | --- |
| **Modifier** | | **Description** | **Try it** |
| public | | The class is accessible by any other class |  |
| *default* | The class is only accessible by classes in the same package. This is used when you don't specify a modifier. You will learn more about packages in the [Packages chapter](https://www.w3schools.com/java/java_packages.asp)  For **attributes, methods and constructors**, you can use the one of the following:   |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Modifier** | | **Description** | **Try it** | | | public | | The code is accessible for all classes |  | | | private | The code is only accessible within the declared class | | |  | | *default* | The code is only accessible in the same package. This is used when you don't specify a modifier. You will learn more about packages in the [Packages chapter](https://www.w3schools.com/java/java_packages.asp) | | |  | | protected | The code is accessible in the same package and **subclasses**. You will learn more about subclasses and superclasses in the [Inheritance chapte](https://www.w3schools.com/java/java_inheritance.asp) | | |  |   **Non-Access Modifiers**  For **classes**, you can use either final or abstract:   |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Modifier** | | **Description** | **Try it** | | | final | | The class cannot be inherited by other classes (You will learn more about inheritance in the [Inheritance chapter](https://www.w3schools.com/java/java_inheritance.asp)) |  | | | abstract | The class cannot be used to create objects (To access an abstract class, it must be inherited from another class. You will learn more about inheritance and abstraction in the [Inheritance](https://www.w3schools.com/java/java_inheritance.asp) and [Abstraction](https://www.w3schools.com/java/java_abstract.asp) chapters) | | |  |   For **attributes and methods**, you can use the one of the following:   |  |  | | --- | --- | | **Modifier** | **Description** | | final | Attributes and methods cannot be overridden/modified | | static | Attributes and methods belongs to the class, rather than an object | | abstract | Can only be used in an abstract class, and can only be used on methods. The method does not have a body, for example **abstract void run();**. The body is provided by the subclass (inherited from). You will learn more about inheritance and abstraction in the [Inheritance](https://www.w3schools.com/java/java_inheritance.asp) and [Abstraction](https://www.w3schools.com/java/java_abstract.asp) chapters | | transient | Attributes and methods are skipped when serializing the object containing them | | synchronized | Methods can only be accessed by one thread at a time | | volatile | The value of an attribute is not cached thread-locally, and is always read from the "main memory" |   **Final**  If you don't want the ability to override existing attribute values, declare attributes as final:  **Static**  A static method means that it can be accessed without creating an object of the class, unlike public:  **Abstract**  An abstract method belongs to an abstract class, and it does not have a body. The body is provided by the subclass: | | | |

* + 1. Java Buzzword.

1. [Simple](https://www.javatpoint.com/features-of-java#Simple)
2. [Object-Oriented](https://www.javatpoint.com/features-of-java#Object-Oriented)
3. [Portable](https://www.javatpoint.com/features-of-java#Portable)
4. [Platform independent](https://www.javatpoint.com/features-of-java#Platform-independent)
5. [Secured](https://www.javatpoint.com/features-of-java#Secured)
6. [Robust](https://www.javatpoint.com/features-of-java#Robust)
7. [Architecture neutral](https://www.javatpoint.com/features-of-java#Architecture-neutral)
8. [Interpreted](https://www.javatpoint.com/features-of-java#Interpreted)
9. [High Performance](https://www.javatpoint.com/features-of-java#High-Performance)
10. [Multithreaded](https://www.javatpoint.com/features-of-java#Multithreaded)
11. [Distributed](https://www.javatpoint.com/features-of-java#Distributed)
12. [Dynamic](https://www.javatpoint.com/features-of-java#Dynamic)
13. **Components of java**

 **Java Virtual Machine (JVM)**

 **Java Runtime Environment (JRE)**

 **Java Development Kit (JDK**

1. **Meaning of main method.**

public static void main(String[] args){

// some code

}

**7. Java Virtual Thread.**

**Virtual Threads in Java**

**Last Updated : 15 Nov, 2023**

**In Java, Virtual threads are now supported by the Java Platform. Virtual threads are lightweight threads that greatly minimize the effort required to create, operate, and manage high volumes systems that are concurrent. As a result, they are more efficient and scalable than standard platform threads.**

**A thread is the smallest processing unit that can be scheduled. It operates concurrently with, and mostly independently of other units of this type. It’s an instance of java.lang.Thread.**

**There are two kinds of threads, platform threads and virtual threads:**

* **Platform Threads**
* **Virtual Threads**

**1. Platform Threads**

**Operating system (OS) threads are implemented with a platform thread acting as a thin wrapper around them. Java code is executed by a platform thread on its parent OS thread, and the platform thread captures its OS thread for the length of its lifetime.**

**As a result, there can only be an equal number of OS threads and platform threads.**

**The operating system usually maintains a large thread stack and additional resources for platform threads. They may be in short supply, but they are suitable for executing many kinds of work.**

**2. Virtual Threads**

**A virtual thread is an instance of java.lang.Thread, independent of any OS thread, is used to run programs. The Java runtime suspends the virtual thread until it resumes when the code calls a blocked I/O operation. Virtual threads have a limited call stack and can only execute one HTTP client call or JDBC query. They are suitable for delayed operations, but not for extended CPU-intensive tasks.**

**Syntax of Virtual Threads:**

**Thread virtualThread = Thread.ofVirtual().start(() -> {  
 // Code to be executed by the virtual thread  
});**

**Why Use Virtual Thread?**

**In your high quantities concurrent applications, use virtual threads if you have many concurrent processes that take a long time to finish. Server applications often handle large numbers of client requests, which requires blocking I/O tasks such as resource access. This makes server applications high-throughput.**

**Virtual threads do not execute code more quickly than platform threads. Their goal is to provide scale (greater throughput) rather than speed (lower latency).**

**Advantages of Java Virtual Threads:**

**There are various advantages to using virtual threads:**

* **Increases the availability of applications**
* **Enhances application throughput.**
* **Reduces the occurrence of ‘OutOfMemoryError: Unable to Create New Native Thread’.**
* **Reduces the amount of memory used by the application**
* **Enhances code quality**
* **Platform Threads are completely compatible with them.**

**How To Create A Virtual Thread?**

**Platform and virtual thread creation is possible with the Thread and Thread.Builder APIs. The methods to build an ExecutorService that launches a new virtual thread for each operation are also defined in the java.util.concurrent.Executors class.**

**1. Using the Thread Class and the Thread.Builder Interface to Create a Virtual Thread**

**Create a version of *Thread.Builder* using the *Thread.ofVirtual()* function to build virtual threads.**

**The following example creates and opens a virtual thread that outputs a message. To wait for the virtual thread to finish its work, it calls the join method. You can see the written message before the main thread ends by doing this.**

**Thread thread = Thread.ofVirtual().start(() -> System.out.println("Hello"));  
thread.join();**

**You can create threads with standard Thread properties, such as the thread name, using *Thread.Builder*. While *Thread.Builder.OfVirtual* creates virtual threads, *Thread.Builder.OfPlatform* creates platform threads.**

**The given code starts and opens a virtual thread called “MyThread,” prints a message while it’s operating, and then uses *t.join()* to wait for the thread to finish. You can follow these steps to a similar program:**

**Entry point method.**

**ava main() Method – public static void main(String[] args)**

**Last Updated : 19 Mar, 2024**

**Java’s main() method is the starting point from where the JVM starts the execution of a Java program. JVM will not execute the code, if the program is missing the main method. Hence, it is one of the most important methods of Java, and having a proper understanding of it is very important.**

**The Java compiler or JVM looks for the main method when it starts executing a Java program. The signature of the main method needs to be in a specific way for the JVM to recognize that method as its entry point. If we change the signature of the method, the program compiles but does not execute.**

**The execution of the Java program, the java.exeis called. The Java.exe in turn makes Java Native Interface or JNI calls, and they load the JVM. The java.exe parses the command line, generates a new String array, and invokes the main() method. By default, the main thread is always a non-daemon thread.**

**Syntax of main() Method**

**Syntax of the main() method is always written as:**

**ava main() Method – public static void main(String[] args)**

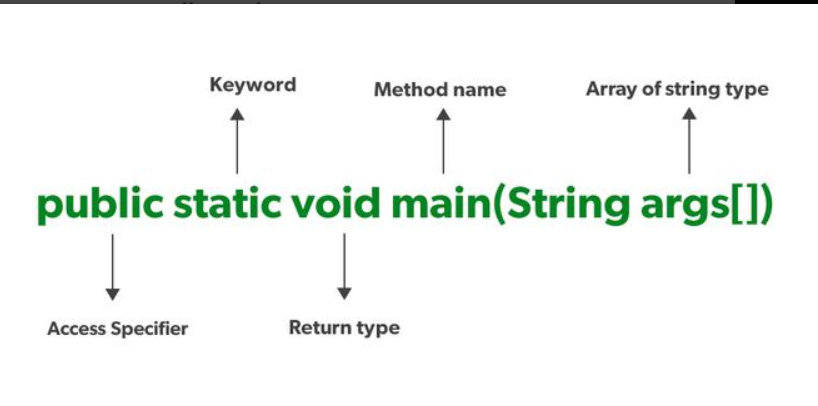
**Last Updated : 19 Mar, 2024**

**Java’s main() method is the starting point from where the JVM starts the execution of a Java program. JVM will not execute the code, if the program is missing the main method. Hence, it is one of the most important methods of Java, and having a proper understanding of it is very important.**

**The Java compiler or JVM looks for the main method when it starts executing a Java program. The signature of the main method needs to be in a specific way for the JVM to recognize that method as its entry point. If we change the signature of the method, the program compiles but does not execute.**

**The execution of the Java program, the java.exeis called. The Java.exe in turn makes Java Native Interface or JNI calls, and they load the JVM. The java.exe parses the command line, generates a new String array, and invokes the main() method. By default, the main thread is always a non-daemon thread.**

**Syntax of main() Method**



|  |
| --- |
|  |

**9. Meaning of System in and System out.**

**Java has 3 streams called *System.in*, *System.out*, and *System.err* which are commonly used to provide input to, and output from Java applications. Most commonly used is probably System.out for writing output to the console from console programs (command line applications).**

**System.in, System.out and System.err are initialized by the Java runtime when a Java VM starts up, so you don't have to instantiate any streams yourself (although you can exchange them at runtime). I will explain each of these streams in deeper detail later in this tutorial.**

**System.in**

**System.in is an** [**InputStream**](https://jenkov.com/tutorials/java-io/inputstream.html) **which is typically connected to keyboard input of console programs. In other words, if you start a Java application from the command line, and you type something on the keyboard while the CLI console (or terminal) has focus, the keyboard input can typically be read via System.in from inside that Java application. However, it is only keyboard input directed to that Java application (the console / terminnal that started the application) which can be read via System.in. Keyboard input for other applications cannot be read via System.in .**

**System.in is not used as often since data is commonly passed to a command line Java application via command line arguments, files, or possibly via network connections if the application is designed for that. In applications with GUI the input to the application is given via the GUI. This is a separate input mechanism from System.in.**

**System.out**

**System.out is a** [**PrintStream**](https://jenkov.com/tutorials/java-io/printstream.html) **to which you can write characters. System.out normally outputs the data you write to it to the CLI console / terminal. System.out is often used from console-only programs like command line tools as a way to display the result of their execution to the user. This is also often used to print debug statements of from a program (though it may arguably not be the best way to get debug info out of a program).**

**System.err**

**System.err is a** [**PrintStream**](https://jenkov.com/tutorials/java-io/printstream.html)**. System.err works like System.out except it is normally only used to output error texts. Some programs (like Eclipse) will show the output to System.err in red text, to make it more obvious that it is error text.**

**Simple System.out + System.err Example:**

**Here is a simple example that uses System.out and System.err:**

**try {**

**InputStream input = new FileInputStream("c:\\data\\...");**

**System.out.println("File opened...");**

**} catch (IOException e){**

**System.err.println("File opening failed:");**

**e.printStackTrace();**

**}**

**Exchanging System Streams**

**Even if the 3 System streams are static members of the java.lang.System class, and are pre-instantiated at JVM startup, you can change what streams to use for each of them. Just set a new InputStream for System.in or a new OutputStream for System.out or System.err, and all further data will be read / written to the new stream.**

**To set a new System stream, use one of th emethods System.setIn(), System.setOut() or System.setErr(). Here is a simple example:**

**OutputStream output = new FileOutputStream("c:\\data\\system.out.txt");**

**PrintStream printOut = new PrintStream(output);**

**System.setOut(printOut);**

**10. Size of data types.**

**Primitive Data Types**

**A primitive data type specifies the size and type of variable values, and it has no additional methods.**

**There are eight primitive data types in Java:**

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Size** | **Description** |
| **byte** | **1 byte** | **Stores whole numbers from -128 to 127** |
| **short** | **2 bytes** | **Stores whole numbers from -32,768 to 32,767** |
| **int** | **4 bytes** | **Stores whole numbers from -2,147,483,648 to 2,147,483,647** |
| **long** | **8 bytes** | **Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807** |
| **float** | **4 bytes** | **Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits** |
| **double** | **8 bytes** | **Stores fractional numbers. Sufficient for storing 15 decimal digits** |
| **boolean** | **1 bit** | **Stores true or false values** |
| **char** | **2 bytes** | **Stores a single character/letter or ASCII values** |

**11. Loader in java.**

**ClassLoader in Java**

**Last Updated : 11 Jul, 2024**

**The Java ClassLoader is an integral part of the** [**Java Runtime Environment**](https://www.geeksforgeeks.org/differences-jdk-jre-jvm/) **(JRE) that dynamically loads Java classes into the** [**Java Virtual Machine**](https://www.geeksforgeeks.org/jvm-works-jvm-architecture/) **(JVM). The Java run time system does not need to know about files and file systems because of classloaders.** [**Java classes**](https://www.geeksforgeeks.org/classes-objects-java/) **aren’t loaded into memory all at once, but when required by an application. At this point, the Java ClassLoader is called by the JRE, and these ClassLoaders load classes into memory dynamically.**

**ClassLoaders play a crucial role in Java’s ability to dynamically load classes into memory as needed, enabling flexibility and efficiency in Java applications.**

**Types of ClassLoaders in Java**

**Java’s ClassLoaders are categorized into different types, each responsible for loading classes from specific locations:**

**1. Bootstrap ClassLoader (Primordial ClassLoader):**

* **The Bootstrap ClassLoader is a machine code responsible for initiating the JVM’s operations.**
* **In Java versions up to 8, it loaded core Java files from rt.jar. However, starting from Java 9, it loads core Java files from the Java Runtime Image (JRT).**
* **Bootstrap ClassLoader operates independently without any parent ClassLoaders.**

**2. Platform Class Loader (Extension ClassLoader):**

* **In Java versions before Java 9, there was an Extension ClassLoader, but from Java 9 onwards, it’s referred to as the Platform Class Loader.**
* **It loads platform-specific extensions from the JDK’s module system.**
* **Platform Class Loader loads files from the Java runtime image or from any other module specified by the system property java.platform or –module-path.**

**3. System ClassLoader (Application ClassLoader):**

* **Also known as the Application ClassLoader, it loads classes from the application’s classpath.**
* **It is a child of the Platform Class Loader.**
* **Classes are loaded from directories specified by the environment variable CLASSPATH, the -classpath or -cp command-line option.**

**12. Garbage collector**

**How Does Garbage Collection in Java works?**

**Java garbage collection is an automatic process. Automatic garbage collection is the process of looking at heap memory, identifying which objects are in use and which are not, and deleting the unused objects. An in-use object, or a referenced object, means that some part of your program still maintains a pointer to that object. An unused or unreferenced object is no longer referenced by any part of your program. So the memory used by an unreferenced object can be reclaimed. The programmer does not need to mark objects to be deleted explicitly. The garbage collection implementation lives in the JVM.**

**Types of Activities in Java Garbage Collection**

**Two types of garbage collection activity usually happen in Java. These are:**

1. **Minor or incremental Garbage Collection: It is said to have occurred when unreachable objects in the young generation heap memory are removed.**
2. **Major or Full Garbage Collection: It is said to have occurred when the objects that survived the minor garbage collection are copied into the old generation or permanent generation heap memory are removed. When compared to the young generation, garbage collection happens less frequently in the old generation.**

**Important Concepts Related to Garbage Collection in Java**

**.**

**13. ⁠Assignments**