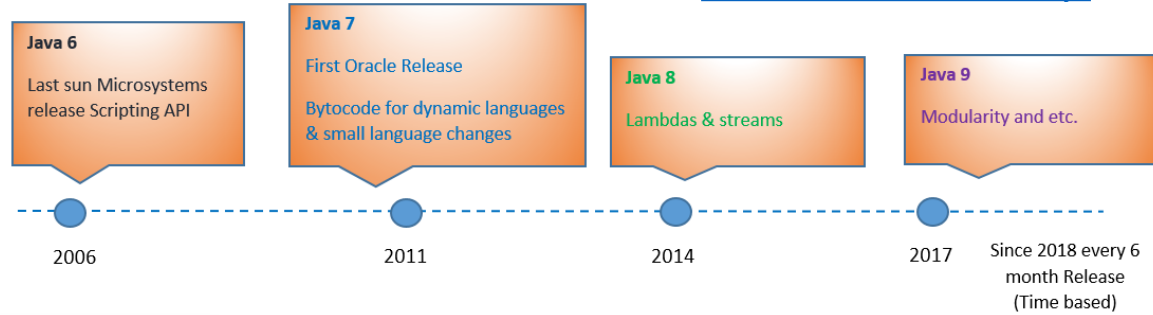


Feature based releases

## Java version history



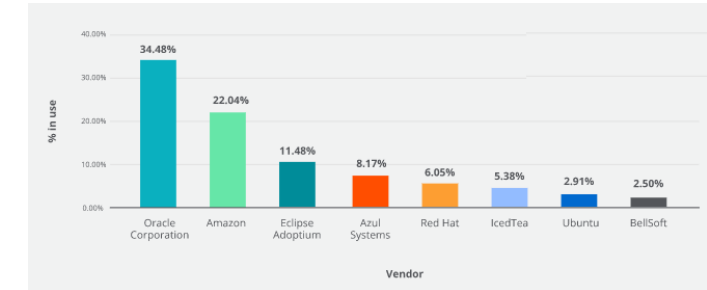
**Adoption** statistics (appr.) in production

in 2020: 84.48% Java **8**, 11.11% Java **11**, in 2022: 46.45% Java **8**, 48% Java **11**

In 2023: Java 17 slightly started to be used in production

Also, note that Java **14** is the most popular non-LTS version

> *java -version*, > *java -fullversion*



# Java Features

<https://github.com/azatsatklichov/Java-Features>

<http://sahet.net/htm/java.html>

[https://en.wikipedia.org/wiki/Java\\_\(programming\\_language\)](https://en.wikipedia.org/wiki/Java_(programming_language))

**Azat Satklichov**  
[azats@seznam.cz](mailto:azats@seznam.cz)

# Sun Microsystems and Oracle

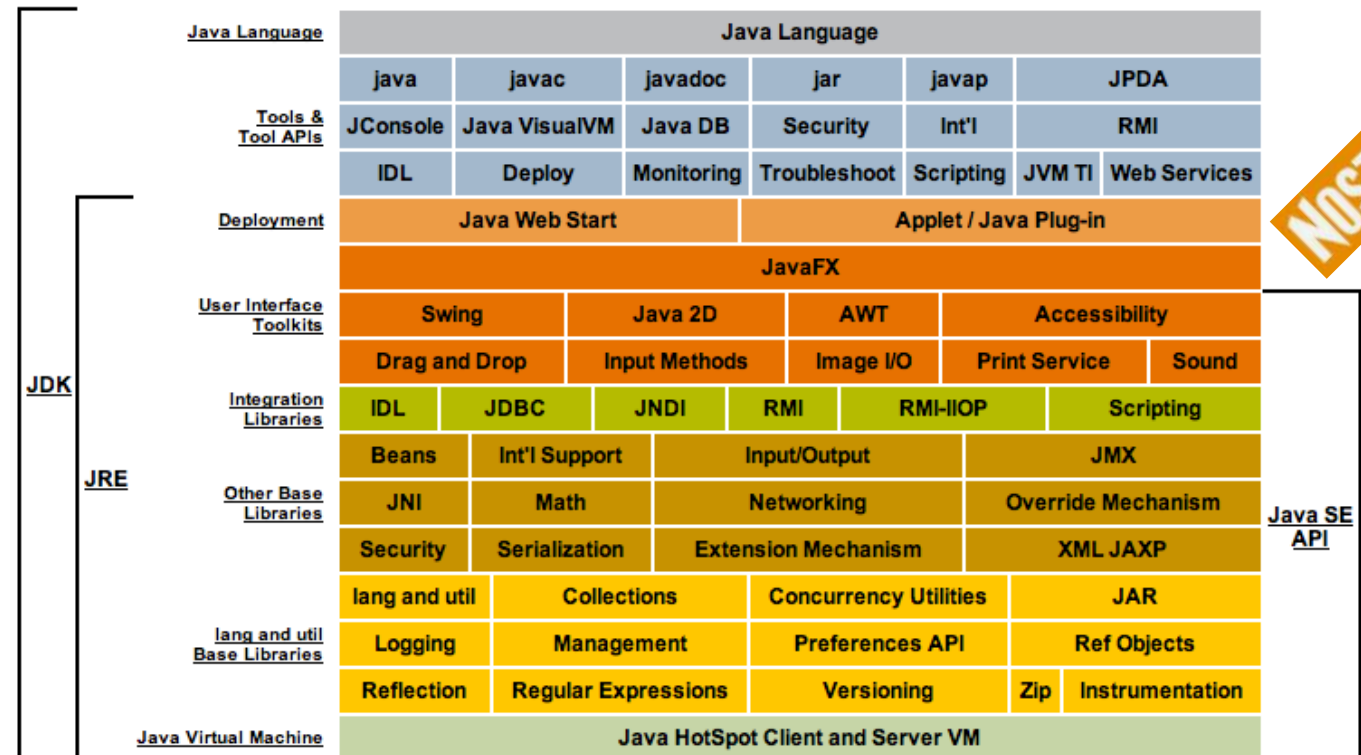


- Java 6 - Last Sun Microsystems release 2006 (Scripting API)
- 27-Jan 2010 Oracle acquired Sun Microsystems - [https://en.wikipedia.org/wiki/Sun\\_Microsystems](https://en.wikipedia.org/wiki/Sun_Microsystems)
- First Oracle release 2011, Java 7 - Bytecode for dynamic languages and some small language changes
- 2014 New Era for Java started with Java 8 release – Lambdas, Streams, new Date Time API, and many more

*Oak* was the initial name for Java given after a big oak tree growing outside James Gosling's window. It went by the name *Green* later, and was eventually named **Java** inspired from Java Coffee, consumed in large quantities by Gosling.



Duke, Oak's smart agent that would later become the Java mascot



NOSTALGIE

# JAVA 8 and Beyond

**JAVA 10 – Last free version of JDK**

**Goal:** convergence of Oracle & OpenJDK codebases

Oracle JDK 8,9,10  
Binary Code License  
Paid  
Oracle Support Contract



Open JDK 8,9, 10  
GPL v2  
Free  
Other option: Amazon Corretto, AdoptOpen JDK (Prebuilt OpenJDK), RedHat Open JDK, ..

Since version 11, **Oracle contributes some commercial features to OpenJDK community**, such as [Flight Recorder](#), [Java Mission Control](#), and [Application Class-Data Sharing](#), as well as the [Z Garbage Collector](#), are now available in OpenJDK. Therefore, **Oracle JDK and OpenJDK builds are essentially identical from Java 11 onward.**

Oracle JDK 11  
Binary Code License  
Paid  
**LTS Support** by Java SE Subscription



Open 11  
GPL v2  
Free  
**LTS Support** by Amazon Corretto, AdoptOpen JDK (Prebuilt OpenJDK), RedHat Open JDK ..

## Main differences between Oracle JDK 11 (A) and Open JDK 11 (B)

- **(A)** [emits a warning](#) when using the `-XX:+UnlockCommercialFeatures` option, with **(B)** builds this option results in an [error](#)
- **(A)** offers a configuration to provide usage log data to the "Advanced Management Console" tool
- **(A)** required third party cryptographic providers to be signed by a known certificate, while cryptography framework in **(B)** has an open cryptographic interface, which means there is no restriction as to which providers can be used
- **(A)** will continue to include installers, branding, and JRE packaging, whereas **(B)** builds are currently available as `zip` and `tar.gz` files
- The `javac -release` command behaves differently for the Java 9 and 10 targets due to the presence of some additional modules in **(A)**'s release
- The output of the `java -version` and `java -fullversion` commands will distinguish Oracle's builds from OpenJDK builds

# JAVA 8 and Beyond

Since Java SE 10, **every six months there** is a new release. Not all releases will be the Long-Term-Support (LTS) releases, it will happen only in every three years. Java SE 17 is the latest LTS version

## Oracle JDK

Contains more tools than the standalone JRE as well as the other components needed for developing Java applications. Oracle strongly recommends using the term JDK to refer to the Java SE (Standard Edition) Development Kit

## Oracle JDK vs. OpenJDK

**Release Schedule:** Oracle will deliver releases every three years (err. ???), while OpenJDK will be released **every six months**. Oracle provides long term support for its releases. On the other hand, OpenJDK supports the changes to a release only until the next version is released.

WoooW LTS 2 years –!!!! - Oracle is proposing that the next LTS release should be **Java 21, Sep. 2023**, which will change the ongoing LTS release cadence from three years to two years, **so in every 2 years after Java 17**.

**Licenses:** Oracle JDK is under Oracle Binary Code License Agreement, whereas OpenJDK has the GNU GPL version 2 with a linking exception.

**Performance:** Even though no real technical difference exists in these two, **Oracle JDK is much better regarding responsiveness and JVM performance**. It puts more focus on stability due to the importance it gives to its enterprise customers. OpenJDK, in contrast, will deliver releases more often. As a result, we can encounter problems with instability. Based on community feedback, we know some OpenJDK users have encountered performance issues.

**Features:** Besides standard features and options, Oracle JDK provides **Flight Recorder, Java Mission Control, and Application Class-Data Sharing features**, [contributed to OpenJDK since Java 11] while OpenJDK has the **Font Renderer feature**. Also, Oracle has more Garbage Collection options and better renderers.

**Development and Popularity:** Oracle JDK is fully developed by Oracle, whereas the OpenJDK is developed by Oracle, OpenJDK, and the Java Community. However, the top-notch companies like Red Hat, Azul Systems, IBM, Apple Inc., SAP AG also take an active part in its development.

When it comes to the popularity with the top companies that use Java Development Kits in their tools, such as Android Studio or IntelliJ IDEA, the Oracle JDK used to be more preferred, but both of them have switched to the OpenJDK based JetBrains builds.

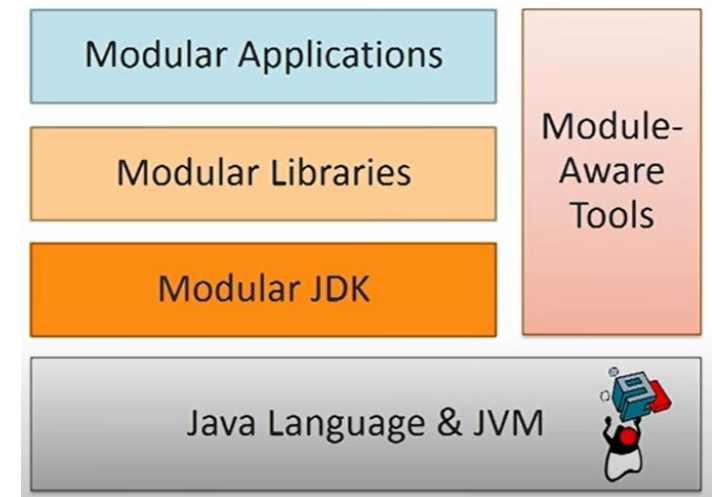
On the other hand, major Linux distributions (Fedora, Ubuntu, Red Hat Enterprise Linux) provide OpenJDK as the default Java SE implementation.

## Open JDK

OpenJDK is a **free and open-source** implementation of the Java SE Platform Edition. Initially, it was based only on the JDK 7. But, since Java 10, the open-source reference implementation of the Java SE platform is the responsibility of the JDK Project. E.g AtoptOpen JDK (Prebuilt OpenJDK), Amazon Corretto, RedHat Open JDK .

# New Features in JAVA 8 and Beyond, <https://github.com/azatsatklichov/Java-Features>

- Functional Interfaces, Lambdas Expressions & Method references
- Stream API, New Date-Time API (migrate from java.util.Date -> java.time)
- Process API
- Optional, Collection Factory Methods
- Modular JDK – biggest change ever made to Java (Language, Compiler, VM, Tooling)
- Local var,
- HTTP Client
- Docker Awarene -JVMs are now aware of being run in a Docker container
- New Performance profile and Security features, TLS 1.3, .. updates
- Continious JVM Improvements: G1 GC, ZGC, Shebandoah, ..
- Tools: FlightRecorder, Java Mission Control, jshell, jDeps, jpackager, jlink, ..
- Language Improvements, Deprecations & Removals
- TLS 1.3, ..
- Records, Switch Expressions, Text Blocks, ..



**Source code:** <https://github.com/azatsatklichov/Java-Features/tree/master/src/main/java/features/in/java8>

# When Migrating Java to Beyond Java 8

When migration consider 3 points: Modularity (**optional**: still classpath based app can be used **--illegal-access=permit**), Library additions, Security&Performance improvements like Docker awareness, G1 GC, etc. see below

- Build tools: Maven min. version 3.8.0 , <configuration> <release>11</release> </configuration> , Gradle version 5 , or Docker [image: maven:3-openjdk-11](#)
- Plugins, .. Eclipse or IntelliJ newer version
- Modularity (internally) – to be prepared to next 20 years
- Deprecated APIs (Base64, SecurityConst, enterprise APIs-CORBA, JAXB,JTA, so use Jakarta EE)
- Removed ones
- Mainly you need to solve, ne to deal with “**encapsulated types / jdk internals**” and “**non-default**” module usage
- Use [jdeps](#) to finds dependencies, to migrate .. E.g. [[> jdeps --jdkinternals MainDemo.class](#) ]
- Try run on Java 11 with: [--illegal-access=deny](#) (not dependent on jsinternals, e.g. **sun.misc.Unsafe**) – **FUTURE PROOF** code for modularity
- For temporary solution (not full migration) use [--add-exports .. --add-modules](#) to solve non-default module , ...
- Start using Java 11 features like List.of(), etc.
- LTS to LST is good practice e.g. Java 11 -> Java 17 -> Java 23. But you can use non-LST
- So still aware of non-LST versions (Java 13,14,15,16, 17(next LST), 18, ..), what is new, removed, deprecated, ..
- Plan your FUTURE : [--illegal-access=deny](#)
- In case to deep dive in JVM, use [visualvm](#) to with installing plugins [visualgc](#) and [grallvm](#)
- So, simply CLASSPATH to MODULEPATH
- Also note that releasing LTS to LTS is a big gap, a lot features, deprecations/removals may impact, better approach is try upgrading non-LTS versions on extra-dev branch or use Amazon/RedHat/OpenJDK LTS versions for those not supported LTS by Oracle

# New Features in JAVA 14

## Features

- 305:[Pattern Matching for instanceof \(Preview\)](#)
- 343:[Packaging Tool \(Incubator\)](#)
- 345:[NUMA-Aware Memory Allocation for G1](#)
- 349:[JFR Event Streaming](#)
- 352:[Non-Volatile Mapped Byte Buffers](#)
- 358:[Helpful NullPointerExceptions](#)
- 359:[Records \(Preview\)](#)
- 361:[Switch Expressions \(Standard\)](#)
- 362:[Deprecate the Solaris and SPARC Ports](#)
- 363:[Remove the Concurrent Mark Sweep \(CMS\) Garbage Collector](#)
- 364:[ZGC on macOS](#)
- 365:[ZGC on Windows](#)
- 366:[Deprecate the ParallelScavenge + SerialOld GC Combination](#)
- 367:[Remove the Pack200 Tools and API](#)
- 368:[Text Blocks \(Second Preview\)](#)
- 370:[Foreign-Memory Access API \(Incubator\)](#)

**Java 14 All Features:** <https://github.com/azatsatklichov/Java-Features>

**Source code:** <https://github.com/azatsatklichov/Java-Features/tree/master/src/main/java/features/in/java14>



# New Features in JAVA 14

Language API Updates, [read more Inside Java](#). E.g. **Project Amber** [pattern matching, text blocks, switch expressions]

- **Removed (since long time clean-up)** - `java.security.acl`, `java.util.jar (Pack200.{Packer, Unpacker})`
- **Deprecated for removal in one of future release** - `Thread::suspend`, `Thread::resume`
- **Deprecated** - [Solaris and SPARC ports](#). Goal: Remove all src-code & build support (maintaining) to these Architectures. **Prj. Valhalla**(reduce memory & comp. overhead.), **Loom**(Fibers..), and **Panama**(foreign memory API,..) require significant changes to CPU-architecture and operating-system specific code.
- **PrintStream** – **new** added methods **write**(byte[] buf) throws IOException, same but no ex. **writeBytes**(byte[] buf)
- **@java.io.Serial, -Xlint:serial** : `@Serial private static final long serialVersionUID = 63L; // motivated from @Override`

Informative NullPointerException Exceptions in Java 14 **Before:** Exception in thread "main" java.lang.NullPointerException at ClassName+line

- **x().y().z()** – just says NPE&line no info no hint, **workaround** – assign it to separate variable //faced in project
- **java -XX:+ShowCodeDetailsInExceptionMessages** Example
- `var a = null; a.b = 1; //Exception in thread "main" java.lang.NullPointerException: Cannot assign field "b" because "a" is null`
- `var a = new Object[][] { null }; a[0][1] = new Object(); //... Cannot store to object array because "a[0]" is null because "a" is null`
- **Switch Expressions** (now **standardized**, since Java 12,13 was preview, was keeps improving and getting feedbacks)
- Able to write switch expression via mixing old-and-new ways. But avoid mixing both see ERR

```
String monthName = switch (monthNumber) {  
    case 1: yield "January";  
    case 2: yield "February"; // rest of cases  
    default: yield "Unknown";  
};
```

```
String monthName;  
switch (monthNumber) {  
    case 1 -> monthName = "January";  
    case 2 -> monthName = "February";  
}
```

```
String monthName;  
switch (monthNumber) {  
    case 1 -> monthName = "January";  
    case 2 -> monthName = "February";  
    case 3: monthName = "March"; //ERR  
}
```



# New Features in JAVA 14

```
if (! (object instanceof BigDecimal b)) {  
    return b.precision();  
}
```

**Preview Feature – Experimental** (like incubator modules), Opt: **--enable-preview --release 14**

- **Pattern Matching** - Eliminates casts. Note: be careful with negation, flow scoping only active when pattern matches.
- instanceof, cast and bind variable in one line.

```
if (object instanceof BigDecimal) {  
    BigDecimal b = (BigDecimal) object;  
    return b.precision();  
}
```

```
if (object instanceof BigDecimal b) { return  
    b.precision(); }
```

```
//future direction, for more functionality usages  
int value = switch (object) {  
    case BigDecimal b -> b.intValue();  
    case String s -> Integer.parseInt(s);  
    case Integer i -> i;  
}
```

- **Text Blocks (multiline)** – Second Preview feature since Java 13

added two more new escape sequences:

- 1) `\<end-of-line>` suppresses the line termination. Compiler not preserve a new line in resulting string.
- 2) `\s` is translated into a single space.

- **Packaging Tools** (Java Web Start & pack200 removed in Java 11), **no GUI just CMD** - create platform-specific **native installers/packages**. Win (MSI/EXE, but [WiX 3.0 or later needed](#)), MacOS (PKG/DMG), Linux (DEB/PRM). Use `jpackage` in each platform to create that platform specific package. Jpackage is based on `java packager` was a JavaFX app.

`jpackage -i . -n JPackagingTools --main-jar hello.jar --main-class JPackagingTools --type exe | deb | dmg --java-options '--enable-preview'`

- **Records** – only contains **DATA** ;)
  - A record class is a shallowly immutable, transparent carrier for a fixed set of values, called the *record components*.
  - Components, constructor, equals/hashCode/toString, **accessors** without getX (way but just via **name()** )

## Java 14 [preview] RECORD (standard part of Java 16)

**Records** are **immutable data classes** that require only the type and name of fields. Good choice when modeling things like domain model classes (via ORM), or DTOs. E.g. Data holders: TypeScript property, or data classes in **Kotlin** or **Scala**.

Using records – with their compiler-generated methods – we can reduce boilerplate code and improve the reliability of our immutable classes. Records are a *semantic* feature; they are **nominal tuples (data carriers)** also tied later to **Sealed classes**.

As with regular Java classes, **we can also include static variables and methods in our records**.

```
AdamRec adam2 = new AdamRec("Adam", "Mary");           System.out.println(adam2.name());  
                                                         AdamRec[name=Adam, address=Mary]
```

### API changes

`java.lang.Class` has two new methods related to record classes for Reflection usage:

- `RecordComponent[] getRecordComponents()`
- `boolean isRecord()`

**Constructor:** public constructor is generated for us. Also can be **customized** (e.g. for validation purposes), or **default** one.

- **The CANONICAL Constructor of a Record Class** – repeats the record class's **components** in the signature
- **The COMPACT Constructor of a Record Class** – to prevent repeating the record class's **components** in the signature.

Used for example to validate fields — only **unchecked** exceptions can be thrown

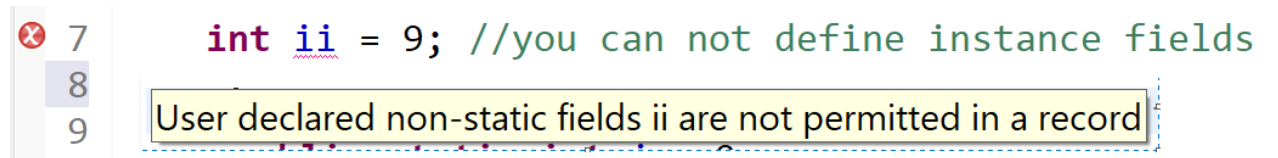
## Java 14 [preview] RECORD (standard part of Java 16)

- The `equals()`, `hashCode()`, and `toString()`, and `private`, `final` fields, and `public` constructor, are generated by the Java compiler.
- **Getters, no setters:** public getters methods – names match the name of our field – for free. (like **TS style:** property)

You can explicitly declare any of the members derived from the header, such as the **public accessor methods** that correspond to the record class's components

```
public String name() {  
    System.out.println("RECORD name is" + name);  
    return name;  
}
```

- `equals()`, `hashCode()`, `toString()`: generated. **Overriding** is possible..
- **Static variables and static and instance Methods:** can be included in records as like in regular classes **BUT**, user declared **non-static fields** XYZ are not permitted in a record. In Enum OK.



```
7 int ii = 9; //you can not define instance fields  
8  
9
```

User declared non-static fields ii are not permitted in a record

- Records are **not** by default **serializable**, but can implement `java.io.Serializable` marker interface (...)
  - Can records replace lombok? **No**. Record's can't provide mutable Object. E.g. `@Data` in Lombok, Builder, etc. ...
- Compiler generated methods already exist in JAVA Enum e.g.: `values()`, `valueOf()`
- **Usages:** DTO (non JPA Entities, Hibernate), Jackson support for Records, Compound Map Keys, Multiple Return value (Obj)

# How does RECORD Look Under the Hood?

Record is just a class with the purpose of holding and exposing data. Record is implicitly final

```
> javac AdamRec.java
```

```
> javap -v -p AdamRec.class //disassembles
```

```
Class<Person> clazz = Person.class;
if (clazz.isRecord()) {
    RecordComponent[] components = clazz.getRecordComponents();
    System.out.println(Arrays.toString(components));
}
```

1. The class is marked **final** (no sub.): public final class net.sahet.record.AdamRec extends **abstract java.lang.Record** (like as Enum)

1. You can implement interfaces, as like Enums. Records are not Serializable by default, but Enums yes.

1. Can not **have native methods**. With native methods compiler can not guarantee manipulating state of record.

2. Two private final fields: private final java.lang.String **name**; private final java.lang.String **address**;

3. generated public constructor: public net.sahet.record.**AdamRec**(java.lang.String, java.lang.String);

4. Two **getters (accessors)**: public java.lang.String **name()**; public java.lang.String **address()**;

5. Also generated: toString(), hashCode(), and equals() which are rely on invokedynamic [**Indy**].

6. Also see [java.lang.ObjectMethods.BootstrapMethods](#)

# New Features in JAVA 14

## JVM Improvements

- [NUMA aware](#) memory allocation for G1 - improve G1 performance on large machines: **-XX:+UseNUMA** .

**Before:** **-XX:+UseParallelGC, NUMA-aware** & helped to improve the performance on single JVM across multiple sockets.

- **ZGC** (since Java 11) [pause time under 10 ms, maybe in future 1 ms] Scale to multi-terabyte heaps, **Colored Pointers**
  - Before Linux/x64 only, now supports **Win&MacOS**

- Concurrent Mark Sweep (**CMS**) GC is **removed**:  
**-XX:+UseConcMarkSweepGC**

```
OpenJDK 64-Bit Server VM warning: Ignoring
option UseConcMarkSweepGC; support was
removed in 14.0
```

- [Deprecate](#) the **ParallelScavenge** + **SerialOld** GC Combination: Due to less use and high maintenance effort, Java 14 deprecates the combination of the parallel young generation and serial old generation GC algorithms.

# New Features in JAVA 14

## Other Improvements

**Non-Volatile Mapped Byte Buffers** (JEP-352) - Improved **FileChannel** API to create **MappedByteBuffer** instances that refer to non-volatile memory (persistent). The primary goal of this JEP is to ensure that clients can access and update NVM from a Java program efficiently and coherently. **Other APIs, or frameworks will be build on top of NVMBB, e.g. Intel's libpmem library**

**Foreign-Memory Access API** (Incubator, JEP-370) – project **PANAMA**, integrating Java with native-env, allow Java API to access **foreign memory** (off-heap memory avoids GC) outside of the Java heap. E.g. used **by JFR. JNI for using** (OpenSSL, V8, Cuda,..)  
- Othercases (**Native mem-alloc**: Metaspace, Threads, CodeCache, GC, Symbols, Native Byte Buffers, Additional Tuning Flag )

There were **two main ways to access native memory** in Java: **ByteBuffer** [HeapByteBuffer and DirectByteBuffer] and **Unsafe**

1) **java.nio.ByteBuffer** [limitations: buffer size max 2GB, GC is resp. for memory deallocation, incorrect usage mem-leak]

2) **sun.misc.Unsafe** [extremely efficient due to its addressing model but may crash the JVM due to illegal memory usage, also now Java 9 jdk.unsupported module ... ]

3) Or using other **libraries like**: **Aeron**, **mapDB**, **memcached**, **ignite**, **Lucene**, and **Netty's ByteBuf** API

4) **Need for a New API (Foreign Memory API)**: Project **PANAMA is born (easier, faster, safer:** accessing the foreign memory[before ByteBuffer used], and invoking the foreign code[before JNI used]) - close to the efficiency of *Unsafe* The foreign memory access API provides a supported, safe, and efficient API to access both heap and. **JEPs in PANAMA:**

**Foreign-Memory Access, Foreign Linker API, Vector API.** New API is built upon three main abstractions: native memory

**MemorySegment** – models a contiguous region of memory

**MemoryAddress** – a location in a memory segment

**MemoryLayout** – a way to define the layout of a memory segment in a language-neutral fashion

- **Java Flight Recorder Event Streaming** (JEP-349) – monitoring events. Now **no need to dump to file**, it can be streamed to get real-time monitoring info. So, instantly we aware what is happening in JVM rather than waiting a dump file.

# New Features in JAVA 15

## Features

- 339: [Edwards-Curve Digital Signature Algorithm EdDSA](#)
- 360: [Sealed Classes \(Preview\)](#)
- 371: [Hidden Classes](#)
- 372: [Remove the Nashorn JavaScript Engine](#)
- 373: [Reimplement the Legacy DatagramSocket API](#)
- 374: [Disable and Deprecate Biased Locking](#)
- 375: [Pattern Matching for instanceof \(Second Preview\)](#)
- 377: [ZGC: A Scalable Low-Latency Garbage Collector](#)
- 378: [Text Blocks](#)
- 379: [Shenandoah: A Low-Pause-Time Garbage Collector](#)
- 381: [Remove the Solaris and SPARC Ports](#)
- 383: [Foreign-Memory Access API \(Second Incubator\)](#)
- 384: [Records \(Second Preview\)](#)
- 385: [Deprecate RMI Activation for Removal](#)

**Java 15 All Features:** <https://github.com/azatsatklichov/Java-Features>

**Source code:** <https://github.com/azatsatklichov/Java-Features/tree/master/src/main/java/features/in/java15>



# New Features in JAVA 15

## Deprecations and Removals

- **Removed JDK ports** - Solaris, SPARC: Solaris/SPARC, Solaris/x64, and Linux/SPARC ports and now it is officially removed
- **RMI deprecations** - `java.rmi.activation`, and the tool '`rmid`' is also deprecated, but tool '`rmic`' is removed
- **Disable and Deprecate Biased Locking** – optimization technique used in Hotspot VM to reduce overhead on uncontended locking. Prior to JDK 15, biased locking is always enabled and available.

**enabled:** `-XX:+UseBiasedLocking -XX:BiasedLockingStartupDelay=0`    **disabled:** `-XX:-UseBiasedLocking`



- **Nashorn JS Engine (old one was Rhino)** – removed, also the tool '`jjs`' is removed. **Alternative** see: **GraalVM**, [Project Detroit](#) (V8)

This JEP also removed the below two modules: `jdk.scripting.nashorn.shell` – contains the `jjs` tool and `jdk.scripting.nashorn` – contains the `jdk.nashorn.api.scripting` and `jdk.nashorn.api.tree` packages.

**JDK has no Javascript-engine anymore.**

```
C:\Users\as892333>jjs
Warning: The jjs tool is planned to be removed from a future JDK release
jjs>
```

**See GraalVM DEMO for more detail in separate presentation - GraalVM**



# New Features in JAVA 15



[GraalVM](#) is an umbrella term, consists of: **Graal** the JIT compiler, and **Truffle** (lang. runtimes), **Substrate VM** (Native Image).

**GraalVM Enterprise** (based on **Oracle JDK**) and **GraalVM Community** (on **OpenJDK**) editions, supports **Java 8, 11, 16**. **Releases:** 1-release GraalVM 19.0 is based on top of JDK version 8u212, latest-21.2.0.1

**History:** GraalVM has its roots in the [Maxine Virtual Machine](#) project at Sun Microsystems Laboratories (now [Oracle Labs](#)). The **goal was** removing dependency to C++ & its problems ... & written in modular, maintainable and extendable fashion in Java itself.

## Project goals

- To improve the performance of [Java virtual machine](#)-based languages to match the performance of native languages.
- To reduce the startup time of JVM-based applications by compiling them ahead-of-time with GraalVM Native Image technology.
- To enable GraalVM integration into the OracleDB, OpenJDK, Node.js, Android/iOS, and to support similar custom embeddings.
- To allow freeform mixing of code from any programming language in a single program, billed as "[polyglot](#) applications".
- To include an easily extended set of "[polyglot programming tools](#)".

# New Features in JAVA 15

## Second Preview Features, since Java 14

### ■ Pattern Matching

```
if (object instanceof BigDecimal) {  
    BigDecimal b = (BigDecimal) object;  
    return b.precision();  
}
```

```
if (object instanceof BigDecimal b) { return  
    b.precision(); }
```

```
int value = switch (object) {  
    case BigDecimal b -> b.intValue();  
    case String s -> Integer.parseInt(s);  
    case Integer i -> i;  
}
```

### ■ Record Enhancements

- A record class is a shallowly immutable, transparent carrier for a fixed set of values, called the *record components*.
- Components, constructor, equals/hashCode/toString, accessors without getX() but just via name()
- This JEP **enhanced the records** with features like support **sealed types, local records, annotation on records, and Reflection APIs for records, generic record class**.
- Also local interfaces and enums also allowed to declare locally in method in Java 15:

```
record Triangle<C extends Coordinate> (C top, C left, C right) {}
```

```
record Customer(String i) implements Billable { }
```

```
record Rectangle(@GreaterThanZero double length, @GreaterThanZero double width) {}
```

```
public void myMethod() {  
    record Point(int x, int y) {} //implicitly static  
    Point p = new Point(1, 1); //  
}
```

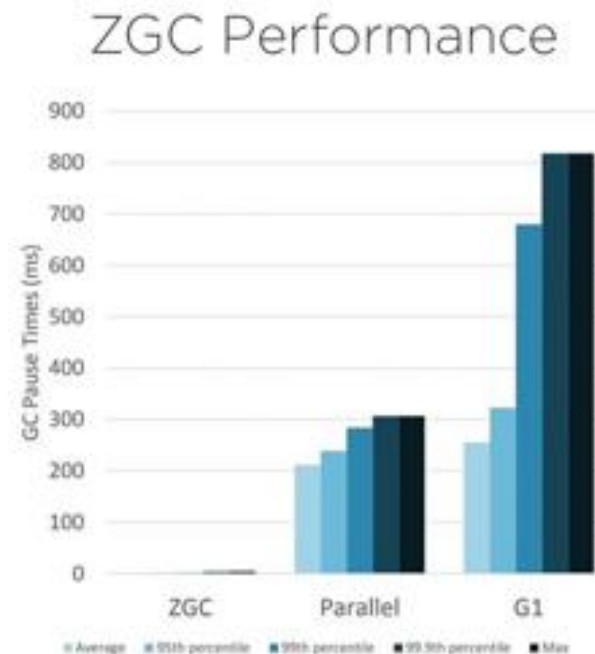
Serializable Records – not by default like Enums. You can serialize and deserialize instances of records, but you can't customize the process by providing writeObject, readObject, readObjectNoData, writeExternal, or readExternal methods.

```
record RangeRecord(int lo, int hi) implements Serializable {} //no need to add serialVersionUID
```

# New Features in JAVA 15

## JVM Improvements

- **ZGC** (since Java 11, .. 14) – **now in Java 15 can be used in production, `-XX:+UseZGC`**
- **Shenandoah** [Since Java 12] Developed by Red Hat, contributed to OpenJDK Backported to Java 8, 11 . `-XX:+UseShenandoahGC` //only supported, RedHat OpenJDK **Became product feature** in Java 15. Oracle JDK and Open JDK has not this feature.
- **No more:** `java -XX:+UnlockExperimentalVMOptions -XX:+UseShenandoahGC`
- Just set `-XX:+UseShenandoahGC` to enable the Shenandoah GC, RedHat JDK has this.



## Other Improvements

- **Foreign-Memory Access** API (Second Incubator, JEP-383 since Java 14) made some changes to the APIs, and it will be still in incubator modules. [new VarHandle API](#), enhanced support for *mapped* memory segments, support parallel processing
- **Hidden classes** (JEP-371) – VM level feature. **Sealed classes JEP-360** app-level feature
- **Edwards-Curve** Digital Signature Algorithm (**EdDSA**) (JEP-339). VM level cryptography. More secure and faster
- Re-Implemented the Legacy **DatagramSocket** API (JEP-373) ([java.net.DatagramSocket](#), [MulticastSocket APIs](#))  
See **Java 13** for Re-implementation of Socket API - [java.net.Socket](#), [ServerSocket](#),  
[made ready for new Java Concurrency models](#) - [see project Loom](#)

# New Features in JAVA 15

- [Sealed Classes](#) (Preview) – Immutable classes are too strong (final can not be inherited), inheritance is too open

ADDS more POWER to MODULAR Programming. Modular programming itself enriched microservices to another level.

```
//Before - nice workaround only via subclasses
public abstract class Option<T> {

    private Option() {}

    public final static class Some<T> extends Option<T> { ... }

    public final static class Empty extends Option<Void> { ... }
    // ...
}
```

```
//with Sealed classes – permits
public sealed class Option<T> permits Some, Empty {
    // ..
}

public final class Some extends Option<T> {
    // ..
} //or non-sealed, sealed

public final class Empty extends Option<Void> {
    // ..
} //or non-sealed, sealed
```

```
//or in same src-file, permits can be omitted
public sealed class Option<T> {

    public final static class Some<T> extends Option<T> {} //or non-sealed, sealed

    public final static class Empty extends Option<Void> {} //or non-sealed, sealed

}

//Record, Enum classes are implicitly final
```

## Constraints on Permitted Subclasses

- They must be accessible by the sealed class at compile time.
- They must directly extend the sealed class
- They must have exactly one of the modifiers: **final, sealed or non-sealed**
- They must be in the same module as sealed class or in the same package(un-named module)

## API change

```
// Sealed classes are also supported by the reflection API, where two public methods have been added to the java.lang.Class
- java.lang.constant.ClassDesc[] permittedSubclasses()
- boolean isSealed()
```

# New Features in JAVA 15

**Hidden Classes (JEP-371)** – This feature is introduced as an alternative to [Unsafe API](#) (`sun.misc.Unsafe`) which isn't recommended outside JDK. **Hidden classes** feature is helpful for anyone who works with dynamic bytecode or JVM languages.

**Hidden classes** (not discoverable and have a limited lifecycle (shorter live) have become a standard way to generate **dynamic classes in Java 15**. Dynamically generated classes provide efficiency and flexibility for **low-latency applications**. They're only needed for a limited time. Hidden classes are used by frameworks that generate classes at runtime and use them indirectly via reflection.

After the introduction of [hidden classes](#), `sun.misc.Unsafe::defineAnonymousClass` is deprecated in Java 15, which will be removed in future releases.

**Hidden classes** are classes that cannot be used directly by the bytecode or other classes. Can also be defined as a member of the [access control nest](#) and can be unloaded independently of other classes. [See Java 11](#) – Nest Based Access (used in **Sealed classes**).

```
MethodHandles.Lookup lookup = MethodHandles.lookup();
```

***Lookup::defineHiddenClass*** method creates the hidden class

## [How to Write Hidden Classes](#)

# New Features in JAVA 16

## Features

- 338: [Vect or API \(Incubator\)](#)
- 347: [Enable C++14 Language Features](#)
- 357: [Migrate from Mercurial to Git](#)
- 369: [Migrate to GitHub](#)
- 376: [ZGC: Concurrent Thread-Stack Processing](#)
- 380: [Unix-Domain Socket Channels](#)
- 386: [Alpine Linux Port](#)
- 387: [Elastic Metaspace](#)
- 388: [Windows/AArch64 Port](#)
- 389: [Foreign Linker API \(Incubator\)](#)
- 390: [Warnings for Value-Based Classes](#)
- 392: [Packaging Tool](#)
- 393: [Foreign-Memory Access API \(Third Incubator\)](#)
- 394: [Pattern Matching for instanceof](#)
- 395: [Records](#)
- 396: [Strongly Encapsulate JDK Internals by Default](#)
- 397: [Sealed Classes \(Second Preview\)](#)

### *Java 16 developer features.*

jpackage, records (standard),  
pattern matching (standard),  
sealed types (second preview),  
foreign-memory access APIs (third incubator), foreign  
linker APIs to replace JNI (incubator),  
vector APIs (incubator)

**Source code:** <https://github.com/azatsatklichov/Java-Features/tree/master/src/main/java/features/in/java16>



# New Features in JAVA 16

**Vector API** (jdk.incubator.vector): **Vector** (**SIMD** – vector instruction), **Scalar** (**SISD**) Processors

Java supports auto vectorization (special case of automatic parallelization) to optimize the arithmetic algorithms, which means the Java (JIT compiler, developers have no control of this vector operation conversion,) will transform some scalar operations (one item at a time) into vector operations (multiple items at a time) automatically;

**Goals:** Clear and concise API, Platform agnostic, Graceful degradation. A vector will be represented by the abstract class **Vector<E>**

## Flynn's taxonomy

Single data stream

**SISD · MISD**

Multiple data streams

**SIMD · MIMD**

Here is a simple **scalar computation** over elements of arrays:

```
void scalarComputation(float[] a, float[] b, float[] c) {  
    for (int i = 0; i < a.length; i++) {  
        c[i] = (a[i] * a[i] + b[i] * b[i]) * -1.0f;  
    }  
}
```

## Alternatives

HotSpot's auto-vectorization is an alternative approach, but it would require significant work.

Equivalent vector computation using the **Vector API**

```
static final VectorSpecies<Float> SPECIES = FloatVector.SPECIES_256;  
  
void vectorComputation(float[] a, float[] b, float[] c) {  
    for (int i = 0; i < a.length; i += SPECIES.length()) {  
        var m = SPECIES.indexInRange(i, a.length);  
        // FloatVector va, vb, vc;  
        var va = FloatVector.fromArray(SPECIES, a, i, m);  
        var vb = FloatVector.fromArray(SPECIES, b, i, m);  
        var vc = va.mul(va).  
            add(vb.mul(vb)).  
            neg();  
        vc.intoArray(c, i, m);  
    }  
}
```

[http://daniel-strecker.com/blog/2020-01-14\\_auto\\_vectorization\\_in\\_java/](http://daniel-strecker.com/blog/2020-01-14_auto_vectorization_in_java/)

# New Features in JAVA 16

## Enable C++14 Language Features

JDK 15 allows the use of [C++14 language features](#) (August 18, 2014) in JDK C++ source code, and give specific guidance about which of those features may be used in HotSpot code. It was limited to the C++98/03 language standards before.

[JEP 397](#) - [Sealed Classes](#) ([Second Preview](#), in Java 17 becomes standard feature)

- [see Java 15 slide](#)
- **Improvements:** Notion of a contextual keywords, restrict the use of a superclass on anonymous classes and lambda expressions, local, enhance narrowing reference to support future directions in [pattern matching](#)

[JEP 389](#): Foreign Linker Memory Access API (3<sup>rd</sup> Incubator) — better alternative to JNI, [see Java 14 slide](#)

Enables Java code to call or can be called by native code written in other languages like C or C++, replace [Java Native Interface \(JNI\)](#). *P.S This is an incubating feature; need add `--add-modules jdk.incubator.foreign` to compile and run the [Java code](#).*

[JEP 393](#) Foreign-Memory Access API (Third Incubator).

Allows Java API to access the foreign memory outside of the Java heap, such as [memcached](#), [Lucene](#), etc.

- Java 14 [JEP 370](#) introduced Foreign-Memory Access API (Incubator).
- Java 15 [JEP 383](#) introduced Foreign-Memory Access API (Second Incubator).
- Java 17 [JEP 412](#) introduced Foreign Function & Memory API (Incubator).

# New Features in JAVA 16

## Records (Standard feature) - [see Java 15 slide](#)

Java 14 [JEP 359](#), first preview. Java 15 [JEP 384](#), second preview. [JEP 395: Records](#). Java 16, standard feature.

## Pattern Matching (Standard feature) - [see Java 14 slide](#)

Java 14 [JEP 305](#), first preview. Java 15 [JEP 375](#), second preview. Java 16, standard feature.

## API Updates

- **Stream API** – `Collectors.toList()` -> **`toList()`** (first modifiable list, last not) so only shortcut is usable for read-only case. No **`toSet`**. Another improvement **`mapMulti()`** fulfils `flatMap`.
- **Date formatting** – `DateTimeFormatterBuilder`

## [JEP 396](#): Strongly Encapsulate JDK Internals by Default

Java 9 [JEP 261](#) introduced the `--illegal-access` option to control the access of the internal APIs and packages of the JDK. This JEP change the default mode of `--illegal-access` option from permit to deny. With this change, the internal packages and APIs (except the [critical internal APIs – e.g. sun.misc.Unsafe](#)) of the JDK will no longer open by default.

Motivation is to discourage the 3<sup>rd</sup>-party libs, frameworks, and tools from using the internal APIs and packages of the JDK.

## Future of Pattern Matching

### Patterns in Switches

```
static String formatterPatternSwitch(Object o) {  
    return switch (o) {  
        case Integer i -> String.format("int %d", i);  
        case Long l    -> String.format("long %d", l);  
        case Double d  -> String.format("double %f", d);  
        case String s  -> String.format("String %s", s);  
        default        -> o.toString();  
    };  
}
```

### Record Patterns

```
if (o instanceof Point(int x, int y)) {  
    System.out.println(x+y);  
}
```

### Array Patterns

```
if (o instanceof String[] { String s1, String s2, ... }) {  
    System.out.println(s1 + s2);  
}
```

# DateTimeFormatterBuilder

Parse or format the textual name of the day period

Time	→	Locale Data Markup Language	→	Formatted
10:05		morning1		in the morning
				in the evening

```
var formatter = new DateTimeFormatterBuilder()
    .appendDayPeriodText(TextStyle.FULL).toFormatter();
var output =
    formatter.format(LocalDate.of(2021, 6, 1, 19, 30));
```

```
List<String> features =  
    Stream.of("Records", "Pattern Matching", "Sealed Classes")  
        .map(String::toLowerCase)  
        .filter(s -> s.contains(" "))  
        .toList();
```

```
<R> Stream<R> mapMulti(BiConsumer<? super T, ? super Consumer<R>> mapper)
```

```
List<Integer> evenNumbers = Stream.of(1, 2, 3, 4)  
    .flatMap(number -> {  
        if(number % 2 == 0) {  
            return Stream.of(number, number);  
        } else {  
            return Stream.of();  
        }  
    })  
    .toList();
```

```
List<Integer> evenNumbers = Stream.of(1, 2, 3, 4)  
    .<Integer>mapMulti((number, downstream) -> {  
        if(number % 2 == 0) {  
            downstream.accept(number);  
            downstream.accept(number);  
        }  
    })  
    .toList();
```

[2, 2, 4, 4]

# New Features in JAVA 16

## Move to Git and GitHub. [JEP 357](#)

**OpenJDK source code [migrated](#) from Mercurial ;)** to Git – which is more mature & modern, have more tooling support, more available hosting , also developer experience. Size of version control system metadata in Mercurial was too big.

- Migrate all single-repository OpenJDK Projects from Mercurial to Git
- Preserve all version control history, including tags
- Reformat commit messages according to Git best practices
- Port the [jcheck](#), [webrev](#), and [defpath](#) tools to Git
- Create a tool to translate between Mercurial and Git hashes
- [JEP 369](#): Migrate to GitHub – migrate all single-repository OpenJDK Projects to GitHub, including both [JDK feature releases](#) and [JDK update releases](#) for versions 11 and later. Hosting, run jcheck (precommit), ...

## JEP 376: [ZGC](#): Concurrent Thread-Stack Processing

This JEP improves the Z Garbage Collector (ZGC) by moving the ZGC [thread-stack processing](#) from safepoints to a concurrent phase. See Java 11 [JEP 333](#) for introduction to ZGC, and it became product of Java 15 by [JEP 377](#)

- Remove thread-stack processing from ZGC safepoints. - Remove all other per-thread root processing from ZGC safepoints.
- Make stack processing lazy, cooperative, concurrent, and incremental.
- Provide a mechanism by which other HotSpot subsystems can lazily process stacks.



# New Features in JAVA 16

## JEP 390: Warnings for Value-Based Classes

This JEP provides a new warning if we synchronize instances of [value-based classes](#); Java 9 deprecated the primitive wrapper class (value-based) constructors and is now marked for removal.

```
34     Double d = 20.0;|
35 Double is a value-based type which is a discouraged argument for the synchronized statement
36     } // javac warning & HotSpot warning
```

## JEP 392: Packaging Tool

Java 14 [JEP 343](#) introduced an jpackage incubating tool and it remained an incubating tool in Java 15, [see slide 51](#). In Java 16, jpackage tool moved from `jdk.incubator.jpackage` to `jdk.jpackage`, and became a standard or product feature.

> jpackage -h //to see usage

# New Features in JAVA 16

## JEP 380: Unix-Domain Socket Channels

The [Unix-domain sockets](#) are used for inter-process communication (IPC) on the same host, which means exchanging data between processes executing on the same host. This JEP add [Unix-domain \(AF UNIX\) socket](#) support to the existing [SocketChannel](#) and [ServerSocketChannel](#).

New Unix-domain Socket classes or APIs:

- New socket address class, `java.net.UnixDomainSocketAddress`
- New `enum`, `java.net.StandardProtocolFamily.UNIX`

## JEP 386: Alpine Linux Port

This JEP port the JDK to [Alpine Linux](#) and other Linux distributions that use [musl](#) implementation. This JDK port enables Java to run out-of-the-box in Alpine Linux, which benefits those Java-dependent frameworks or tools like Tomcat and Spring. *P.S The Alpine Linux contains small image size, widely adopted in cloud deployments, microservices, and container environments.*

## JEP 387: Elastic Metaspace

Java 8 [JEP 122](#) removed the PermGen, and introduced [Metaspace](#), a native off-heap memory manager in the hotspot. This JEP improves the metaspace memory management by returning unused HotSpot class-metadata or metaspace memory to the operating system more promptly, reducing the metaspace footprint, and simplifying the metaspace code.

## JEP 388: Windows/AArch64 Port

This JEP port the JDK to Windows/AArch64, running JDK + Windows on ARM hardware, server, or ARM-based laptop. P.S The Windows/AArch64 is a popular demand in the end-user market.

# New Features in JAVA 17 – LTS Release

## Features

- 306: [Restore Always-Strict Floating-Point Semantics](#)
- 356: [Enhanced Pseudo-Random Number Generators](#)
- 382: [New macOS Rendering Pipeline](#)
- 391: [macOS/AArch64 Port](#)
- 398: [Deprecate the Applet API for Removal](#)
- 403: [Strongly Encapsulate JDK Internals](#)
- 406: [Pattern Matching for switch \(Preview\)](#)
- 407: [Remove RMI Activation](#)
- 409: [Sealed Classes](#)
- 410: [Remove the Experimental AOT and JIT Compiler](#)
- 411: [Deprecate the Security Manager for Removal](#)
- 412: [Foreign Function & Memory API \(Incubator\)](#)
- 414: [Vector API \(Second Incubator\)](#)
- 415: [Context-Specific Deserialization Filters](#)

<https://openjdk.java.net/projects/jdk/17/>

**Source code:** <https://github.com/azatsatklichov/Java-Features/tree/master/src/main/java/features/in/java17>

# Oracle Releases Java 17

- *Next Java long-term support release, September 14, 2021* (previous LTS was Java 11, 9.2018, impl. 70 JEPs) delivers thousands of performance, stability, and security updates, as well as **14 JEPs** (JDK Enhancement Proposals) that further improve the Java language and platform to help developers be more productive.
- *Oracle JDK 17 gives customers security, performance, and bug-fix updates through September 2029*
- *Oracle is proposing that the next LTS release should be **Java 21** and made available in September **2023**, which will change the ongoing LTS release cadence **from three** years to **two years**.*
- **Accelerating Java's Adoption in the Cloud**  
To accelerate Java adoption in the cloud, Oracle recently introduced the [Oracle Java Management Service](#) (a new Oracle Cloud Infrastructure [OCI](#))

# Updates and Improvements to Libraries

- [JEP 306](#): Restore Always-Strict Floating-Point Semantics
- [JEP 356](#): Enhanced Pseudo-Random Number Generator
- [JEP 382](#): New macOS Rendering Pipeline
- [JEP 391](#): macOS AArch64 Port

# Removals and Deprecations

- [JEP 398](#): Deprecate the Applet API for Removal
- [JEP 407](#): Remove RMI Activation
- [JEP 410](#): Remove the Experimental AOT and JIT Compiler
- [JEP 411](#): Deprecate the Security Manager for Removal

<https://www.oracle.com/emea/news/announcement/oracle-releases-java-17-2021-09-14/>

# New Features, Previews&Incubators in JAVA 17

- [Sealed Classes](#) (Standard feature) – was preview feature since [Java 15](#)

This enhancement is yet another improvement from [Project Amber](#)

- [JEP 403](#): **Strongly Encapsulate JDK Internals**
- [JEP 406](#): [Pattern Matching for switch](#) (Preview) (still preview since [Java 14](#))
- [JEP 412](#): **Foreign Function and Memory API (Incubator)** - [see Java 14](#), [Java 15](#)
- [JEP 414](#): **Vector API (Second Incubator)**

# New Features in JAVA 18

## Features

- 400: [UTF-8 by Default](#)
- 408: [Simple Web Server](#)
- 413: [Code Snippets in Java API Documentation](#)
- 416: [Reimplement Core Reflection with Method Handles](#)
- 417: [Vector API \(Third Incubator\)](#)
- 418: [Internet-Address Resolution SPI](#)
- 419: [Foreign Function & Memory API \(Second Incubator\)](#)
- 420: [Pattern Matching for switch \(Second Preview\)](#)
- 421: [Deprecate Finalization for Removal](#)

<https://openjdk.java.net/projects/jdk/18/>

**Source code:** <https://github.com/azatsatklichov/Java-Features/tree/master/src/main/java/features/in/java17>



# New Features in JAVA 18

- [JEP 400](#): **UTF-8 by Default** - default charset of the standard Java APIs
- [JEP 408](#): **Simple Web Server** - Provide a command-line tool to start a minimal web server (static HTTP server) that serves static files only. No CGI or servlet-like functionality is available.
- [JEP 413](#): **Code Snippets in Java API Documentation** - @snippet tag for JavaDoc's Standard Doclet, to simplify the inclusion of example source code in API documentation
- [JEP 416](#): **Reimplement Core Reflection with Method Handles** - Reimplement java.lang.reflect.Method, Constructor, and Field on top of java.lang.invoke method handles.
- [JEP 417](#): **Vector API (Third Incubator)**

# New Features in JAVA 18

- **[JEP 418](#): Internet-Address Resolution SPI** - define a service-provider interface (SPI) for host name and address resolution, so that [java.net.InetAddress](#) can make use of resolvers other than the platform's built-in resolver
- **[JEP 419](#): Foreign Function & Memory API (Second [Incubator](#))** - [see Java 14](#), [Java 15](#), [Java 17](#)
- **[JEP 420](#): Pattern Matching for switch (Second Preview)** - Enhance the Java programming language with pattern matching for switch expressions and statements, along with extensions to the language of patterns.
- **[JEP 421](#): Deprecate Finalization for Removal** - Finalization remains enabled by default for now, but can be disabled to facilitate early testing
- [S](#)



# THANK YOU

## References

<https://www.baeldung.com/oracle-jdk-vs-openjdk>

<https://medium.com/@javachampions/java-is-still-free-c02aef8c9e04>

<https://www.journaldev.com/13106/java-9-modules>

<https://mkyong.com/java/what-is-new-in-java-15/>

<https://blogs.oracle.com/javamagazine/post/java-project-amber-lambda-loom-panama-valhalla>

<https://docs.oracle.com/en/java/javase/17/language/java-language-changes.html#GUID-67ED83E7-D79F-4F46-AA33-41031E5CD094>