Java 9, 10, 11... N

Grzegorz Piwowarek

@pivovarit

Java 9

JEP 102: Process API Updates

Improve the API for controlling and managing operating-system processes.

New methods in Process class leveraging CompletableFuture and Streams

JEP 110: HTTP/2 Client (Incubator)

Define a new HTTP client API that implements HTTP/2 and WebSocket, and can replace the legacy HttpURLConnection

JEP 143: Improve Contended Locking

Field reordering and cache line alignment

Speed up PlatformEvent::unpark()

Fast Java monitor enter operations

Fast Java monitor exit operations

Fast Java monitor notify/notifyAll operations

JEP 158: Unified JVM Logging

Introduce a common logging system for all components of the JVM.

-Xlog:gc=info*

JEP 193: Variable Handles

Define a standard means to invoke the equivalents of various java.util.concurrent.atomic and sun.misc.Unsafe operations upon object fields and array elements

JEP 213: Milling Project Coin

Allow @SafeVargs on private instance methods

Allow effectively-final variables to be used as resources in the try-with-resources statement

Allow diamond with anonymous classes if the argument type of the inferred type is denotable

Disallow _ as a one-character identifier

JEP 222: jshell: The Java Shell (Read-Eval-Print Loop)

Provide an interactive tool to evaluate declarations, statements, and expressions of the Java programming language, together with an API so that other applications can leverage this functionality.

JEP 223: New Version-String Scheme

Define a version-string scheme that easily distinguishes major, minor, and security-update releases, and apply it to the JDK.

Be easily understandable by humans, and easily parsable by programs

JEP 243: Java-Level JVM Compiler Interface

Develop a Java based JVM compiler interface (JVMCI) enabling a compiler written in Java to be used by the JVM as a dynamic compiler.

JEP 248: Make G1 the Default Garbage Collector

Make G1 the default garbage collector on 32- and 64-bit server configurations.

JEP 254: Compact Strings

Adopt a more space-efficient internal representation for strings.

```
public final class String
  implements ... {

    private final byte[] value; // instead of char[]
}
```

JEP 259: Stack-Walking API

Define an efficient standard API for stack walking that allows easy filtering of, and lazy access to, the information in stack traces.

```
StackWalker instance = StackWalker.getInstance();
```

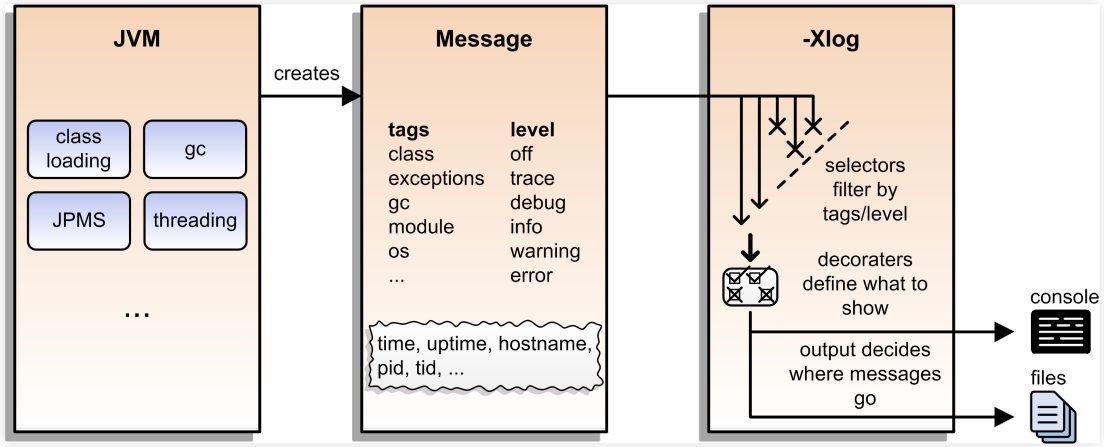
JEP 269: Convenience Factory Methods for Collections

Define library APIs to make it convenient to create instances of collections and maps with small numbers of elements, so as to ease the pain of not having collection literals in the Java programming language.

```
List.of(1, 2, 3);
Set.of(1, 2, 3);
Map.of("key", 42);
```

JEP 271: Unified GC Logging

Reimplement GC logging using the unified JVM logging framework introduced in JEP 158.



https://blog.codefx.org/java/unified-logging-with-the-xlog-option/

JEP 274: Enhanced Method Handles

Enhance the MethodHandle, MethodHandles, and MethodHandles.

JEP 280: Indify String Concatenation

Change the static String-concatenation bytecode sequence generated by javac to use invokedynamic calls to JDK library functions.

JEP 285: Spin-Wait Hints

Define an API to allow Java code to hint that a spin loop is being executed.

JEP 295: Ahead-of-Time Compilation

Compile Java classes to native code prior to launching the virtual machine.

Java 10

JEP 286: Local-Variable Type Inference

Enhance the Java Language to extend type inference to declarations of local variables with initializers.

```
var foo = new User("Jan");
```

JEP 304: Garbage-Collector Interface

Improve the source code isolation of different garbage collectors by introducing a clean garbage collector (GC) interface.

JEP 307: Parallel Full GC for G1

Improve G1 worst-case latencies by making the full GC parallel.

JEP 310: Application Class-Data Sharing

To improve startup and footprint, extend the existing Class-Data Sharing ("CDS") feature to allow application classes to be placed in the shared archive.

We can save about 340MB of RAM for a Java EE app server that includes 6 JVM processes consuming a total of 13GB of RAM (~2GB of that is for class meta data).

JEP 312: Thread-Local Handshakes

Introduce a way to execute a callback on threads without performing a global VM safepoint. Make it both possible and cheap to stop individual threads and not just all threads or none.

JEP 317: Experimental Java-Based JIT Compiler

Enable the Java-based JIT compiler, Graal, to be used as an experimental JIT compiler on the Linux/x64 platform.

JEP 322: Time-Based Release Versioning

Revise the version-string scheme of the Java SE Platform and the JDK, and related versioning information, for present and future time-based release models.

Java 11

JEP 318: Epsilon: A No-Op Garbage Collector

Develop a GC that handles memory allocation but does not implement any actual memory reclamation mechanism

JEP 320: Remove the Java EE and CORBA Modules

Remove the Java EE and CORBA modules from the Java SE Platform and the JDK.

JEP 321: HTTP Client (Standard)

Standardize the incubated HTTP Client API introduced in JDK 9, via JEP 110, and updated in JDK 10.

JEP 323: Local-Variable Syntax for Lambda Parameters

Allow var to be used when declaring the formal parameters of implicitly typed lambda expressions.

JEP 328: Flight Recorder

Provide a low-overhead data collection framework for troubleshooting Java applications and the HotSpot JVM.

```
-XX:+FlightRecorder -XX:FlightRecorderOptions=stackdepth=128
```

JEP 331: Low-Overhead Heap Profiling

Provide a low-overhead way of sampling Java heap allocations, accessible via JVMTI.

JEP 333: ZGC: A Scalable Low-Latency Garbage Collector (Experimental)

The Z Garbage Collector, also known as ZGC, is a scalable low-latency garbage collector.

JEP 335: Deprecate the Nashorn JavaScript Engine

Deprecate the Nashorn JavaScript script engine and APIs, and the jjs tool, with the intent to remove them in a future release.

Java 12

JEP 189: Shenandoah: A Low-Pause-Time Garbage Collector (Experimental)

Add a new garbage collection (GC) algorithm named Shenandoah which reduces GC pause times by doing evacuation work concurrently with the running Java threads. Pause times with Shenandoah are independent of heap size, meaning you will have the same consistent pause times whether your heap is 200 MB or 200 GB.

JEP 230: Microbenchmark Suite

Add a basic suite of microbenchmarks to the JDK source code, and make it easy for developers to run existing microbenchmarks and create new ones.

JEP 325: Switch Expressions (Preview)

Extend the switch statement so that it can be used as either a statement or an expression, and that both forms can use either a "traditional" or "simplified" scoping and control flow behavior.

```
switch (day) {
    case MONDAY, FRIDAY, SUNDAY -> System.out.println(6);
    case TUESDAY                -> System.out.println(7);
    case THURSDAY, SATURDAY     -> System.out.println(8);
    case WEDNESDAY              -> System.out.println(9);
}
```

JEP 334: JVM Constants API

Introduce an API to model nominal descriptions of key class-file and run-time artifacts, in particular constants that are loadable from the constant pool.

Java 13

JEP 350: Dynamic CDS Archives

Extend application class-data sharing to allow the dynamic archiving of classes at the end of Java application execution. The archived classes will include all loaded application classes and library classes that are not present in the default, base-layer CDS archive.

JEP 351: ZGC: Uncommit Unused Memory

ZGC does not currently uncommit and return memory to the operating system, even when that memory has been unused for a long time. This behavior is not optimal for all types of applications and environments, especially those where memory footprint is a concern.

JEP 353: Reimplement the Legacy Socket API

Replace the underlying implementation used by the java.net.Socket and java.net.ServerSocket APIs with a simpler and more modern implementation that is easy to maintain and debug. The new implementation will be easy to adapt to work with user-mode threads, a.k.a. fibers, currently being explored in Project Loom.

JEP 354: Switch Expressions (Preview)

Extend switch so it can be used as either a statement or an expression.

JEP 355: Text Blocks (Preview)

A text block is a multi-line string literal that avoids the need for most escape sequences, automatically formats the string in a predictable way, and gives the developer control over format when desired.

Java 14

JEP 349: JFR Event Streaming

Expose JDK Flight Recorder data for continuous monitoring.

JEP 305: Pattern Matching for instanceof (Preview)

Enhance the Java programming language with pattern matching for the instanceof operator.

JEP 359: Records (Preview)

Enhance the Java programming language with records.

JEP 361: Switch Expressions (Standard)

Extend switch so it can be used as either a statement or an expression

JEP 363: Remove the Concurrent Mark Sweep (CMS) Garbage Collector

Remove the Concurrent Mark Sweep (CMS) garbage collector.

JEP 352: Non-Volatile Mapped Byte Buffers

Add new JDK-specific file mapping modes so that the FileChannel API can be used to create MappedByteBuffer instances that refer to non-volatile memory.

JEP 358: Helpful NullPointerExceptions

Improve the usability of NullPointerExceptions generated by the JVM by describing precisely which variable was null.

Java 15

JEP 378: Text Blocks (Standard)

Add text blocks to the Java language.

JEP 360: Sealed Classes (Preview)

Enhance the Java programming language with sealed classes and interfaces. Sealed classes and interfaces restrict which other classes or interfaces may extend or implement them.

JEP 373: Reimplement the Legacy DatagramSocket API

Replace the underlying implementations of the java.net.DatagramSocket and java.net.MulticastSocket APIs with simpler and more modern implementations that are easy to maintain and debug.

JEP 375: Pattern Matching for instanceof (Second Preview)

Enhance the Java programming language with pattern matching for the instanceof operator.

JEP 377: ZGC: A Scalable Low-Latency Garbage Collector (Production)

Change the Z Garbage Collector from an experimental feature into a product feature.

JEP 379: Shenandoah: A Low-Pause-Time Garbage Collector (Production)

Change the Shenandoah garbage collector from an experimental feature into a product feature.

JEP 384: Records (Second Preview)

Enhance the Java programming language with records, which are classes that act as transparent carriers for immutable data.

JEP 371: Hidden Classes

Introduce hidden classes, which are classes that cannot be used directly by the bytecode of other classes. Hidden classes are intended for use by frameworks that generate classes at run time and use them indirectly, via reflection.

JEP 372: Remove the Nashorn JavaScript Engine

Remove the Nashorn JavaScript script engine and APIs, and the jjs tool.

Java 16

JEP 397: Sealed Classes (Second Preview)

Enhance the Java programming language with sealed classes and interfaces. Sealed classes and interfaces restrict which other classes or interfaces may extend or implement them.

JEP 395: Records(release)

Enhance the Java programming language with records.

JEP 394: Pattern Matching for instanceof

Enhance the Java programming language with pattern matching for the instanceof operator.

JEP 376: ZGC: Concurrent Thread-Stack Processing

Move ZGC thread-stack processing from safepoints to a concurrent phase.

Java 17

JEP 409: Sealed Classes

Enhance the Java programming language with sealed classes and interfaces. Sealed classes and interfaces restrict which other classes or interfaces may extend or implement them.

JEP 410: Remove the Experimental AOT and JIT Compiler

Remove the experimental Java-based ahead-of-time (AOT) and just-in-time (JIT) compiler. This compiler has seen little use since its introduction and the effort required to maintain it is significant.

JEP 411: Deprecate the Security Manager for Removal

Deprecate the Security Manager for removal in a future release. The Security Manager dates from Java 1.0. It has not been the primary means of securing client-side Java code for many years, and it has rarely been used to secure server-side code.

JEP 403: Strongly Encapsulate JDK Internals

Strongly encapsulate all internal elements of the JDK, except for critical internal APIs such as sun.misc.Unsafe. It will no longer be possible to relax the strong encapsulation of internal elements via a single command-line option, as was possible in JDK 9 through JDK 16.