JAVA 12

Java 8 : March 18, 2014
Java 9 : July 27, 2017
Java 10 : March 2018
Java 11 : September 25, 2018
Java 12 : March 19, 2019
Java SE 12 is here and even though it's not an LTS release

Java 12 Open JDK -> Download link Oracle : https://jdk.java.net/12/
Relesae note -> https://jdk.java.net/12/release-notes
Oracle Java 12 ->
https://www.oracle.com/technetwork/java/javase/downloads/jdk12-downloads-5295953.html

Switch Expressions -> 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. These changes will simplify everyday coding, and also
prepare the way for the use of pattern matching (JEP 305) in switch. This will be a preview language feature.

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.

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. Easy to add new benchmarks. Easy to update tests as APIs and options change, are deprecated,
or are removed during development

A way, and a suite of microbenchmarks, to easily test the performance of JDK, based on Java Microbenchmark Harness (JMH) will be added to JDK source code.

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.

API modeling the key class-file and run-time artifacts such as constant pool. Such API will contain classes like ConstantDesc, ClassDesc,
and the draft of this API is available
here:
https://cr.openjdk.java.net/~vromero/constant.api/javadoc.04/java/lang/invoke/constant/package-summary.html.

One AArch64 Port, Not Two -> Remove all of the sources related to the arm64 port while retaining the 32-bit ARM port and the 64-bit aarch64 port.
There are two different set of sources, thus ports, targeting ARM 64-bit in the JDK. One is contributed by Oracle, arm64 (hotspot/cpu/arm),
and the other is aarch64 (hotspot/cpu/aarch64). This JEP removes arm64, thus all source code used with
#ifdefs under hotspot/cpu/arm will be removed and 64-bit ARM build will be default to aarch64. hotspot/cpu/arm will still provide the 32-bit ARM port.

Default CDS Archives -> Enhance the JDK build process to generate a class data-sharing (CDS) archive, using the default class list, on 64-bit platforms.
Class Data-Sharing (CDS) is a feature to reduce startup time and benefit from memory sharing. However, if you do not install the JRE with the installer,
the CDS archive is not generated by default and java -Xshare:dump has to be run manually.

This can be observed in JDK 11. If you install the JDK 11 GA Release from http://jdk.java.net/11/ , lib/server folder does not contain the CDS archive,
classes.jsa file. If you run java -Xshare:dump, it will be generated.

With this JEP, CDS archive will be generated by default
Improve out-of-the-box startup time
Eliminate the need for users to run -Xshare:dump to benefit from CDS

Abortable Mixed Collections for G1 -> Make G1 mixed collections abortable if they might exceed the pause target.Make all pauses in G1 abortable.

Promptly Return Unused Committed Memory from G1 -> Enhance the G1 garbage collector to automatically return Java heap memory to the operating system when idle.
Sharing of committed but empty pages between Java processes. Memory should be returned (uncommitted) to the operating system.
The process of giving back memory does not need to be frugal with CPU resources, nor does it need to be instantaneous.
Use of different methods to return memory other than available uncommit of memory.
Support for other collectors than G1.

```
public class JEP325 {

        public static void main(String[] args) {

  // args[0] is the day of week, starting from 1-sunday
  final int day = Integer.valueOf(args[0]);

  // traditional switch
  switch (day) {
    case 2:
    case 3:
    case 4:
    case 5:
    case 6:
      System.out.println("weekday");
      break;
    case 7:
    case 1:
      System.out.println("weekend");
      break;
    default:
      System.out.println("invalid");
  }

  // case L -> syntax
  // no break necessary, only code next to -> runs
  switch (day) {
    case 2, 3, 4, 5, 6 -> System.out.println("weekday");
    case 7, 1 -> System.out.println("weekend");
    default -> System.out.println("invalid");
  }

  // switch expression
  // then switch should be exhaustive if used as expression
  // break <value_of_switch_expression> syntax for blocks
  final String attr = switch (day) {
    case 2, 3, 4, 5, 6 -> "weekday";
    case 7, 1 -> "weekend";
    // it is possible to do this without a block and break
    // so default -> "invalid"; is actually enough here
    default -> {
      break "invalid";
    }
    };

    System.out.println(attr);

            }

}
```

Raw String Literals (dropped from JDK 12 release)

```
public class JEP326 {

  public static void main(String args[]) {

    // traditional string
    final String s1 = "test";
    // traditional multiline string
    final String s2 = "line1\nline2";

    // raw string literals
    final String rs1 = `test`;
    final String rs2 = `
                        line1
                          line2
                            line3`;
    final String rs3 = ``backtick`inside``;
    final String rs4 = `\n`;

    System.out.println(rs1);
    System.out.println(rs2);
    System.out.println(rs2.align());
    System.out.println(rs3);

    // String::unescape() is not implemented yet on jdk12+21
    System.out.println(rs4.length());
    // System.out.println(rs4.unescape().length());

  }

}
```