Java 14

Java 14 released on March 17, 2020, exactly six months after its previous version as per Java's new release cadence. In this tutorial, we'll look at a summary of new and deprecated features of version 14 of the language. We also have more detailed articles on Java 14 that offer an in-depth view of the new features. A few features have been carried over in Java 14 from the previous version. Let's look at them one by one.

1.. Switch Expressions (JEP 361)

These were first introduced as a preview feature in JDK 12, and even in Java 13, they continued as preview features only. But now, switch expressions have been standardized so that they are part and parcel of the development kit. What this effectively means is that this feature can now be used in production code, and not just in the preview mode to be experimented with by developers. As a simple example, let's consider a scenario where we'd designate days of the week as either weekday or weekend. Prior to this enhancement, we'd have written it as:

```
boolean isTodayHoliday;
switch (day) {
    case "MONDAY":
    case "TUESDAY":
    case "WEDNESDAY":
    case "THURSDAY":
    case "FRIDAY":
        isTodayHoliday = false;
        break;
    case "SATURDAY":
    case "SUNDAY":
        isTodayHoliday = true;
        break;
    default:
        throw new IllegalArgumentException("What's a " + day);
}
```

With switch expressions, we can write the same thing more succinctly:

```
boolean isTodayHoliday = switch (day) {
    case "MONDAY", "TUESDAY", "WEDNESDAY", "THURSDAY", "FRIDAY" -> false;
    case "SATURDAY", "SUNDAY" -> true;
    default -> throw new IllegalArgumentException("What's a " + day);
};
```

2.. Text Blocks (JEP 368)

Text blocks continue their journey to getting a mainstream upgrade and are still available as preview features. In addition to the capabilities from JDK 13 to make multiline strings easier to use, in their second preview, text blocks now have two new escape sequences:

- \: to indicate the end of the line, so that a new line character is not introduced
- \s: to indicate a single space

For example:

```
String multiline = "A quick brown fox jumps over a lazy dog; the lazy dog howls loudly.";
```

can now be written as:

```
String multiline = """
    A quick brown fox jumps over a lazy dog; \
    the lazy dog howls loudly.""";
```

This improves the readability of the sentence for a human eye but does not add a new line after *dog;*.

3.. New Preview Features

3.1 Pattern Matching for instanceof (JEP 305)

JDK 14 has introduced pattern matching for instanceof with the aim of eliminating boilerplate code and make the developer's life a little bit easy. To understand this, let's consider a simple example.

Before this feature, we wrote:

```
if (obj instanceof String) {
    String str = (String) obj;
    int len = str.length();
    // ...
}
```

Now, we don't need as much code to start using obj as String:

```
if (obj instanceof String str) {
    int len = str.length();
    // ...
}
```

In future releases, Java is going to come up with pattern matching for other constructs such as a switch.

3.2. Records (JEP 359)

Records were introduced to reduce repetitive boilerplate code in data model POJOs. They simplify day to day development, improve efficiency and greatly minimize the risk of human error. For example, a data model for a User with an id and password can be simply defined as:

```
public record User(int id, String password) { };
```

As we can see, we are making use of a new keyword, record, here. This simple declaration will automatically add a constructor, getters, equals, hashCode and toString methods for us. Let's see this in action with a JUnit:

```
private User user1 = new User(0, "UserOne");

@Test
public void givenRecord_whenObjInitialized_thenValuesCanBeFetchedWithGetters() {
    assertEquals(0, user1.id());
    assertEquals("UserOne", user1.password());
}

@Test
public void whenRecord_thenEqualsImplemented() {
    User user2 = user1;
    assertTrue(user1, user2);
}

@Test
public void whenRecord_thenToStringImplemented() {
    assertTrue(user1.toString().contains("UserOne"));
}
```

4.. New Production Features

Along with the two new preview features, Java 14 has also shipped a concrete production-ready one.

4.1 Helpful NullPointerExceptions (JEP 358)

Previously, the stack trace for a NullPointerException didn't have much of a story to tell except that some value was null at a given line in a given file. Though useful, this information only suggested a line to debug instead of painting the whole picture for a developer to understand, just by looking at the log. Now Java has made this easier by adding the capability to point out what exactly was null in a given line of code. For example, consider this simple snippet:

```
int[] arr = null;
arr[0] = 1;
```

Earlier, on running this code, the log would say:

```
Exception in thread "main" java.lang.NullPointerException
at com.baeldung.MyClass.main(MyClass.java:27)
```

But now, given the same scenario, the log might say:

```
java.lang.NullPointerException: Cannot store to int array because "a" is null
```

As we can see, now we know precisely which variable caused the exception.

5.. Incubating Features

These are the non-final APIs and tools that the Java team comes up with, and provides us for experimentation. They are different from preview features and are provided as separate modules in the package jdk.incubator.

5.1 Foreign Memory Access API (JEP 370)

This is a new API to allow Java programs to access foreign memory, such as native memory, outside the heap in a safe and efficient manner. Many Java libraries such as mapDB and Memcached do access foreign memory and it was high time the Java API itself offered a cleaner solution. With this intention, the team came up with this JEP as an alternative to its already existing ways to access non-heap memory – ByteBuffer API and sun.misc.Unsafe API. Built upon three main abstractions of **MemorySegment**, **MemoryAddress** and **MemoryLayout**, this API is a safe way to access both heap and non-heap memory.

5.2. Packaging Tool (JEP 343)

Traditionally, to deliver Java code, an application developer would simply send out a JAR file that the user was supposed to run inside their own JVM. However, users rather expected an installer that they'd double click to install the package on their native platforms, such as Windows or macOS. This JEP aims to do precisely that. Developers can use jlink to condense the JDK down to the minimum required modules, and then use this packaging tool to create a lightweight image that can be installed as an exe on Windows or a dmg on a macOS.

6.. JVM/Hotspot Features

6.1 ZGC on Windows (JEP 365) and macOS (JEP 364) – Experimental

The Z Garbage Collector, a scalable, low-latency garbage collector, was first introduced in Java 11 as an experimental feature. But initially, the only supported platform was Linux/x64. After receiving positive feedback on ZGC for Linux, Java 14 has ported its support to Windows and macOS as well. Though still an experimental feature, it's all set to become production-ready in the next JDK release.

6.2 NUMA-Aware Memory Allocation for G1 (JEP 345)

Non-uniform memory access (NUMA) was not implemented so far for the G1 garbage collector, unlike the Parallel collector. Looking at the performance improvement that it offers to run a single JVM across multiple sockets, this JEP was introduced to make the G1 collector NUMA-aware as well. At this point, there's no plan to replicate the same to other HotSpot collectors.

6.3 JFR Event Streaming (JEP 349)

With this enhancement, JDK's flight recorder data is now exposed so that it can be continuously monitored. This involves modifications to the package jdk.jfr.consumer so that users can now read or stream the recording data directly.

7.. Deprecated or Removed Features

Java 14 has deprecated a couple of features:

- Solaris and SPARC Ports (JEP 362) – because this Unix operating system and RISC processor are not in active development since the past few years
- ParallelScavenge + SerialOld GC Combination (JEP 366) – since this is a rarely used combination of GC algorithms, and requires significant maintenance effort

There are a couple of removals as well:

- Concurrent Mark Sweep (CMS) Garbage Collector (JEP 363) – deprecated by Java 9, this GC has been succeeded by G1 as the default GC. Also, there are other more performant alternatives to use now, such as ZGC and Shenandoah, hence the removal
- Pack200 Tools and API (JEP 367) – these were deprecated for removal in Java 11, and now removed