**Java 10 Features**

**Local Variable In type Reference**

We don't provide the data type of *message*. Instead, we mark *the message*as a *var*, and the compiler infers the type of *message*from the type of the initializer present on the right-hand side.

Note

Can not be used as type

List.copyOf(), Set.copyOf(), and Map.copyOf()

With Java 10, we now also have the possibility to create immutable copies of collections. For this purpose, we have the static interface methods List.copyOf(), Set.copyOf() and Map.copyOf().

If we create such a copy and then modify the original collection, the changes will no longer affect the copy:

List<Integer> immutable = List.copyOf(list);

list.add(4);

System.out.println("immutable = " + immutable);

⟶

immutable = [1, 2, 3]

**Optional.orElseThrow()**

Optional, introduced in Java 8, provides the get() method to retrieve the value wrapped by the Optional. Before calling get(), you should always check with isPresent() whether a value exists:

Optional<String> result = getResult();

if (result.isPresent()) {

System.out.println(result.get());

}

If the Optional is empty, get() would otherwise throw a NoSuchElementException.

To minimize the risk of an unintended exception, IDEs and static code analysis tools issue a warning if get() is used without isPresent():

**Java 10 offers a nicer solution with the method or ElseThrow():**

The method is an exact copy of the get() method – only the name is different. Since it is clear from the name that this method can throw an exception, misunderstandings are ruled out. The static code analysis no longer criticizes the usage as a code smell.

Here is the source code of both methods for comparison:

public T get() {

if (value == null) {

throw new NoSuchElementException("No value present");

}

return value;

}

public T orElseThrow() {

if (value == null) {

throw new NoSuchElementException("No value present");

}

return value;

}

After the version format was (finally) changed from the somewhat cryptic 1.8.0\_291 to a much more readable 9.0.4 from Java 8 to 9, JEP 322 added the release date in Java 10 – and for Java 11, an "LTS" (Long-Term Support) in advance.

The command java -version returns the following answers in Java 8 to 11:

**Java 8:**

$ java -version

java version "1.8.0\_291"

Code language: plaintext (plaintext)

Java 9:

$ java -version

java version "9.0.4"

Code language: plaintext (plaintext)

Java 10:

$ java -version

java version "10.0.2" 2018-07-17

**Parallel Full GC for G1**

With JDK 9, the Garbage-First (G1) garbage collector has replaced the parallel collector as the default GC.

While the parallel GC could perform a full garbage collection (i.e., cleaning up all regions of the heap) in parallel with the running application, this was not possible with G1 until now. G1 had to temporarily stop the application ("stop-the-world"), leading to noticeable latencies.

Since G1 was designed to avoid full collections as much as possible, this rarely posed a problem.

In Java 10, with JDK Enhancement Proposal 307, the full gargage collection of the G1 collector has now also been parallelized. The worst-case latencies (pause times) reach those of the parallel collector.

**Experimental Java-Based JIT Compiler**

JIT compiler is written in C++ and is used to convert Java into Byte Code. Now Java 10 has option to enable an experimental Java based JIT compiler, Graal to be used instead of standard JIT compiler. Graal is using JVMCI, JVM Compiler Interface which was introduced in Java 9. Graal is available in Java 9 as well. With Java 10, we can enable Graal to test and debug the experimental JVM compiler.

**Root Certificates**

Cacerts store, prior to Java 10 is an empty set. It is required to contain a set of certificates which can be used to establish trust in certificates chain of various security protocols of vendors.

OpenJDK builds are not having such certificates which is why critical security components like TLS didn't work in default build.

Now as Oracle has open sourced the root certificates using Oracle JAVA SE Root CA program, OpenJDK builds can now have root certificates and thus can reduce the difference between OpenJDK and Oracle JDK.