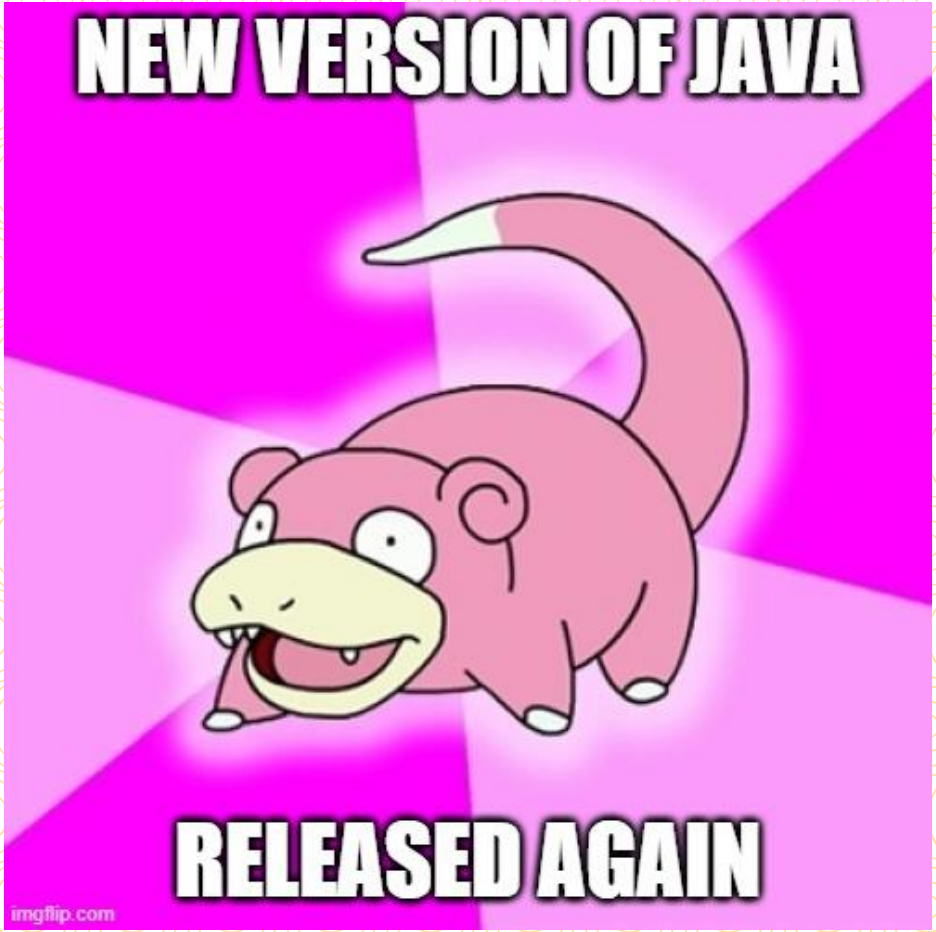
Java 15: what's new



*Release was on the 15th of September, 2020

My previous talk about Java14 can be found here:

https://github.com/akuleshov7/my-conference-presentations/tree/master/Java14

Let’s refresh

Changes in Java distribution

How it affects me?

Any difference?

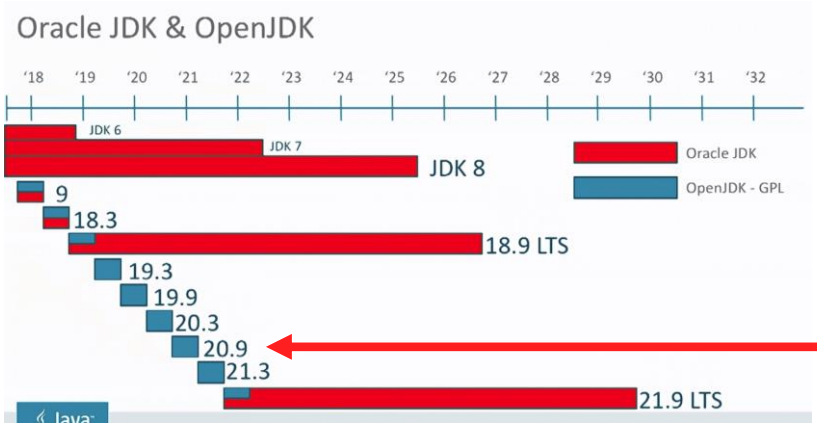
no real technical difference

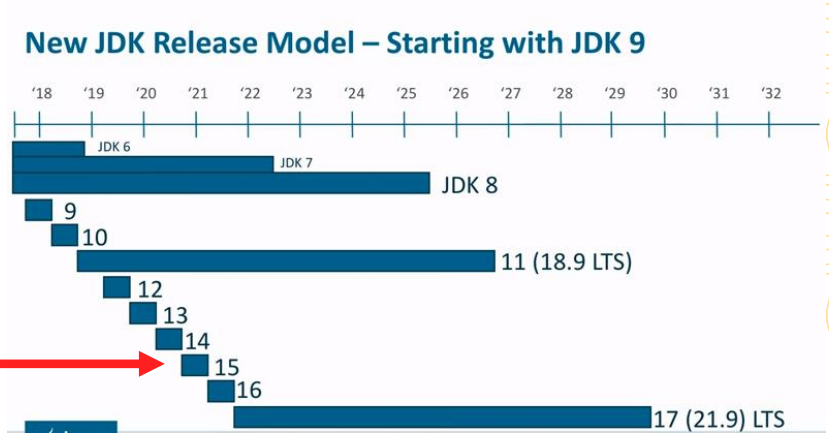
build process for the Oracle JDK is based on OpenJDK.

- Can be used in PROD for free
- Performance is good enough for PROD
- 6 months of support

- Should pay for commercial license
- Performance is better
- LTS (5 years)

Release cycle





Some vocabular

JEP

JDK **E**nhancement **P**roposal

(informal JCP)

JSR

Java **S**pecification **R**equest

Commitee

Best representatives from the best companies. Rotation each year.

JCP

Java **C**ommunity **P**rocess

JEP-12: Preview features

A *preview feature* is a new feature of Java language, that is fully specified, fully implemented, and yet impermanent. Provokes developer feedback based on real world use. Can become permanent in the future.

For example in Java 11 we had experimental options:

-XX:+UnlockExperimentalVMOptions -XX:+UseZGC

Finally!
Let’s move to features of Java15

Brief overview

Disclaimer

My Huawei project **diKTat** was added to **Kotlin-awesome** list, so I will criticize Java and will love Kotlin.

Please forgive me ☺

1

Language features

Features that 99% of you can and will use

JEP-360

Sealed classes!

JEP-360: Sealed classes (Enums on steroids)

- Preview feature **(-enable-preview -release 15**)
- New syntax in language that came from Scala/Kotlin
- Described: https://openjdk.java.net/jeps/360

JEP-360: Sealed classes (Enums on steroids)

- New modifiers for classes and interfaces: “**sealed/non-sealed**”
- New keyword **“permits”**

JEP-360: Sealed classes (Enums on steroids)

- New modifiers for classes and interfaces: “**sealed/non-sealed**”
- New keyword **“permits”**
- Asserts control over which other types may be its subclasses

JEP-360: Sealed classes (Enums on steroids)

- New modifiers for classes and interfaces: “**sealed/non-sealed**”
- New keyword **“permits”**
- Asserts control over which other types may be its subclasses
- A sealed class or interface can be extended or implemented only by those classes and interfaces **permitted** to do so.

JEP-360: Sealed classes (Enums on steroids)

- New modifiers for classes and interfaces: “**sealed/non-sealed**”
- New keyword **“permits”**
- Asserts control over which other types may be its subclasses
- A sealed class or interface can be extended or implemented only by those classes and interfaces **permitted** to do so.
- Extremely needed in switch statements and **pattern matching**
- Works nice with records (data classes)

JEP-360: how it works

```
sealed interface Shape permits Rectangle, Circle  {}
```

JEP-360: how it works

```
sealed interface Shape permits Rectangle, Circle  {}


record Rectangle(int width, int height) implements Shape {}
record Circle(int radius) implements Shape {}
```

JEP-360: limitations

- Sealed class/interface should stay in the same file with inheritors (if **permits** keyword was not used)
- Inherited classes should be only sealed/non-sealed/final

JEP-360: limitations

- Sealed class/interface should stay in the same file with inheritors (if **permits** keyword was not used)
- Inherited classes should be only **sealed/non-sealed/final**
- Inherited classes with permit keyword should stay **in the same module**
- Local classes (anonymous classes) cannot be inherited from sealed

JEP-360: examples

```
public sealed interface Shape permits Rectangle {}
record Rectangle(int width, int height) {} // not implemented interface
```

```
$ java: invalid permits clause (subclass Rectangle must extend sealed class)
```

JEP-360: examples

```
public sealed interface Shape permits Rectangle {}
record Rectangle(int width, int height) {} // not implemented interface
```

$ java: invalid permits clause (subclass Rectangle must extend sealed class)

```
public sealed interface Shape {} // no inheritors/implementations
```

$ java: sealed class must have subclasses

JEP-360: future

- This feature is made to support pattern matching, that is still **not finished**!
- https://openjdk.java.net/jeps/8213076

JEP-360: future

- This feature is made to support pattern matching, that is still **not finished**!
- https://openjdk.java.net/jeps/8213076
- It should work in the following way (like in Scala/Kotlin):

```
sealed interface Color permits BiColor, TriColor { }

record BiColor(int r, int g, int b) implements Color {}
record TriColor(int r, int g, int b) implements Color {}

// …
void foo(Color color) {
    switch (color) {
        case BiColor i -> 0;
        case TriColor j ->1;
    }
}
```

JEP-339

Edwards-Curve Digital Signature Algorithm

JEP-339: Edwards-Curve Digital Signature

- Crypto algorithm of digital signature (standard RFC 8032)
- Description: https://openjdk.java.net/jeps/339
- Security and crypto developers say it is one of the main features of Java15 ☺

JEP-375

Pattern matching (second preview)

JEP-375: Pattern matching for instanceof

- Preview feature
  (-enable-preview -release 15)
- This is just a second preview
- And still it is **not a NORMAL** pattern matching
- Description: http://openjdk.java.net/jeps/375
- New syntax in language, but no changes from Java14.

JEP-375 : Pattern matching for instanceof

Before Java 14

```
Object obj = "Ivan loves instanceof";

if (obj instanceof String) {
    String s = (String) obj;
    System.out.println(s.length());
}
```

JEP-375: Pattern matching for instanceof

After Java 14

```
Object obj = "Ivan loves instanceof";

if (obj instanceof String s){
    System.out.println(s.length());
}
```

JEP-384

Records (second preview)

JEP-384: Records (aka data classes)

- Preview feature
  (-enable-preview -release 15)
- Part of Valhalla Project
- New syntax in language
- Description: https://openjdk.java.net/jeps/384

JEP-384: what has changed from Java14

- Now the JEP explicitly prohibits declaring **native methods** in records.

JEP-384: what has changed from Java14

- Now the JEP explicitly prohibits declaring **native methods** in records.
- The implicitly declared fields corresponding to the record components of a record class are final and **should not be modified via reflection**

JEP-384: what has changed from Java14

- Now the JEP explicitly prohibits declaring **native methods** in records.
- The implicitly declared fields corresponding to the record components of a record class are final and **should not be modified via reflection**
- Records can also be defined **within methods** to store intermediate values. Unlike local classes, a local record is implicitly static.

JEP-384: Records (aka data classes)

```
public record Person(String name, String surname){}
```

- All needed methods are generated by compiler
- Final, cannot be abstract

JEP-384: Records (aka data classes)

```
public final class Person extends java.lang.Record {
    private final java.lang.String name;
    private final String surname;

    public Person(java.lang.String name, String surname) { /* compiled code */ }

    public java.lang.String toString() { /* compiled code */ }

    public final int hashCode() { /* compiled code */ }

    public final boolean equals(java.lang.Object o) { /* compiled code */ }

    public java.lang.String name() { /* compiled code */ }

    public String surname() { /* compiled code */ }
}
```

JEP-378

Text blocks finally released!

JEP-378: Text Blocks (released)

- Added some changes to **JEP-378**: Text Blocks (Preview)
- Will be different from the Java 13 version, but same as in Java 14
- Finally released after second review

JEP-368: Text Blocks (Second Preview)

Before Java 13

```
String html = "<html>\n" +
              "    <body>\n" +
              "        <p>Hello, world</p>\n" +
              "    </body>\n" +
              "</html>\n";
```

JEP-368: Text Blocks (Second Preview)

In Java 13

```
String html = """
              <html>
                  <body>
                      <p>Hello World</p>
                  </body>
              </html>
              """;
```

JEP-368: Text Blocks (Second Preview)

After Java 15 (by the default)

```
String text = """
              Here I want spaces in the end not being trimmed:   \
              Here I will use the second version of this feature:    \s
              This will work? Nice!
              """;
```

- **** - added for newline
- **\s** – added for explicit space

JEP-371

Hidden Classes

JEP-371: Hidden classes

- Hidden classes - cannot be used directly by the bytecode of other classes
- Intended for use by frameworks that generate classes at run time and use them indirectly, via reflection

JEP-371: Hidden classes

- Hidden classes - cannot be used directly by the bytecode of other classes
- Intended for use by frameworks that generate classes at run time and use them indirectly, via reflection
- Change in Java API, no language changes
- Description: https://openjdk.java.net/jeps/371

JEP-371: Hidden classes

- **ClassLoader::defineClass** and **Lookup::defineClass**
- No difference if the class were generated at run time or at compile time

JEP-371: Hidden classes

- **ClassLoader::defineClass** and **Lookup::defineClass**
- No difference if the class were generated at run time or at compile time
- But **Lookup::defineHiddenClass** solves this issue

2

Tools and libs

Features that 10% of you will use,
speedrun

JEP-378

ZGC

JEP-378: Z garbage collector

- Now it is a default GC in Java
- Scalable low latency garbage collector

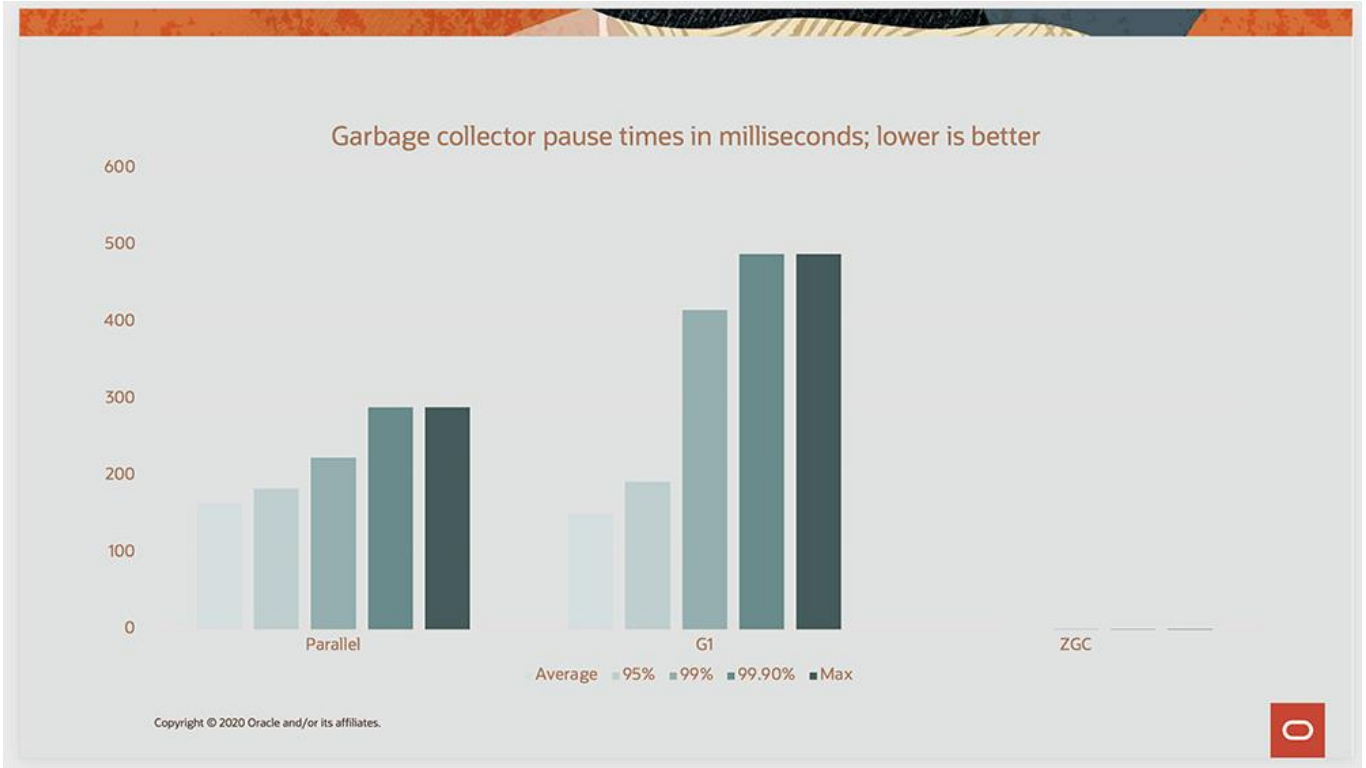
JEP-378: Z garbage collector

- Now it is a default GC in Java
- Scalable low latency garbage collector
- Max pause times of a few milliseconds (stop the world)

JEP-378: Z garbage collector

- Now it is a default GC in Java
- Scalable low latency garbage collector
- Max pause times of a few milliseconds (stop the world)
- Pause times do not increase with the heap or live-set size
- Handle heaps ranging from a 8MB to 16TB in size

JEP-378: Z garbage collector



JEP-379

Shenandoah GC

JEP-379: Shenandoah GC

- Now it is enabled as production feature
- Performing more garbage collection work concurrently with the running Java program

JEP-379: Shenandoah GC

- Now it is enabled as production feature
- Performing more garbage collection work concurrently with the running Java program
- Shenandoah does the bulk of GC work concurrently
- Pause times are no longer directly proportional to the size of the heap
- Created by Russian Java-Champion A. Shipilev

3

Things that you will not even notice

Features that 0% of you will notice

You won’t notice it

- JEP 372: Remove the Nashorn JavaScript Engine
- JEP 374: Disable and Deprecate Biased Locking
- JEP 381: Remove the Solaris and SPARC Ports
- JEP 385: Deprecate RMI Activation for Removal
- JEP 373: Reimplement the Legacy DatagramSocket API

NO

Java language features are still much weaker than **Kotlin** and **Scala**

Thanks!

DO YOU HAVE ANY QUESTIONS?