23. hroug
godišnja konferencija

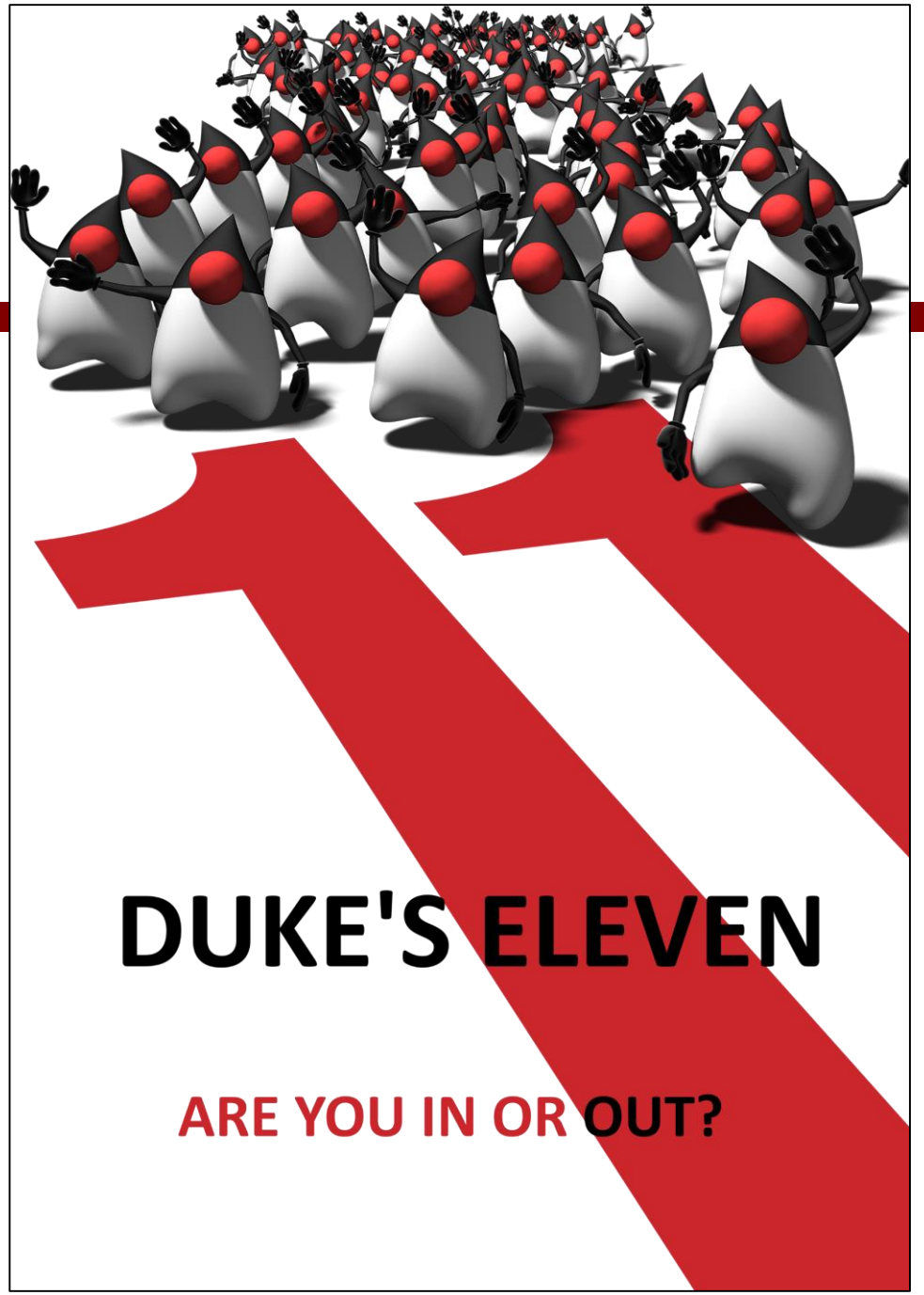
Above and Beyond
Java 9, 10, 11, 12…

dr. sc. **Branko Mihaljević**

and

dr. sc. **Martin Žagar**

HUJAK



A bit of Sentimental Journey

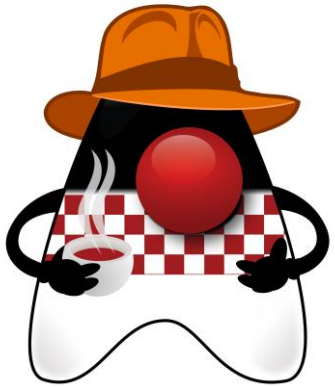
- **HUJAK** founded in **2011**
- Interested in **meetups**
  - Well, more drinkups ☺
- No one dreamed about **organizing conferences** ☺
- However, we've got some **help**… ☺

- ***… the beginning of a beautiful friendship …***

JavaCro conferences – 2012-2018

Javantura conferences – 2014-2018

Javantura



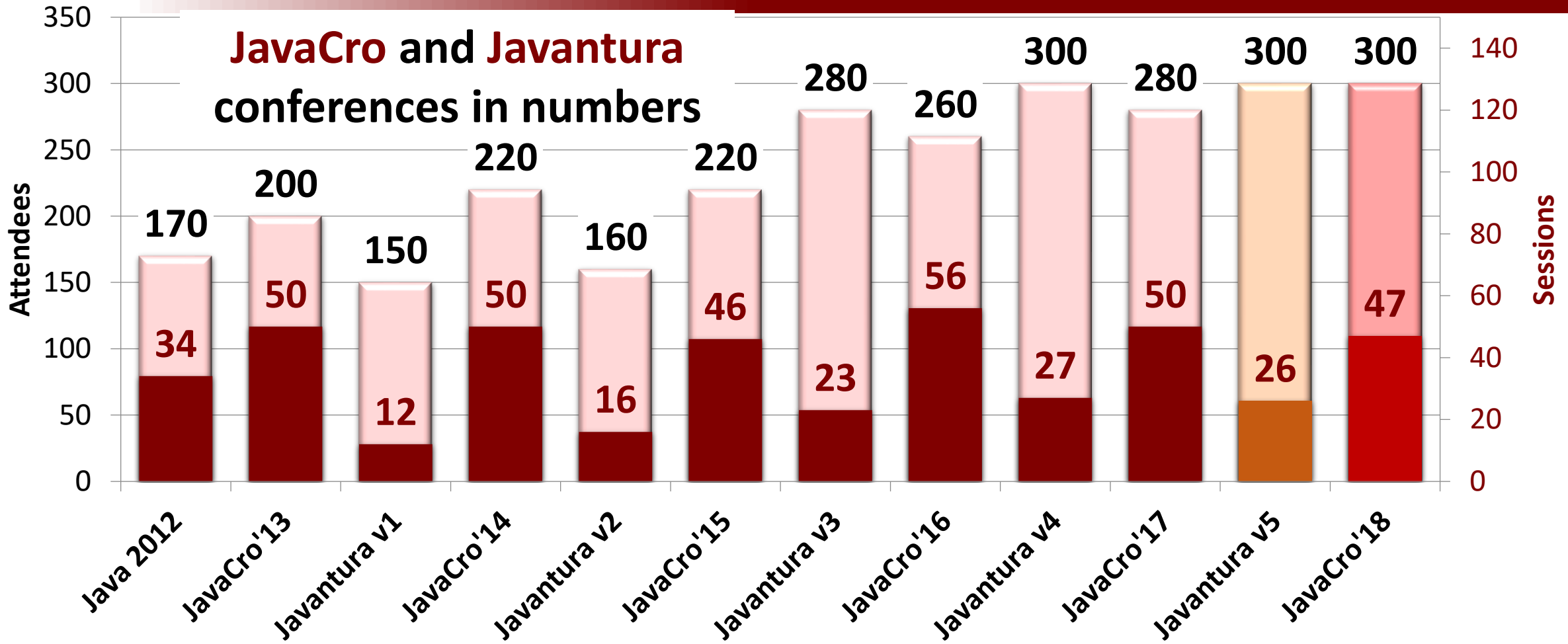
Java-related Conferences in Croatia

Conference	Location	Date	Sessions	Tracks	Attendees	Countries
JavaCro'18	Rovinj	7.-9.5.2018.	47	5	300	15
Javantura v5	Zagreb	17.2.2018.	26	3	300	-
JavaCro'17	Rovinj	10.-12.5.2017.	50	5	280	15
Javantura v4	Zagreb	11.2.2017.	27	3	300	-
HrOUG 2016	Rovinj	18.-22.10.2016.	7 (od 96)	1 (od 9)	450	11
JavaCro'16	Rovinj	18.-20.5.2016.	56	5	260	15
Javantura v3	Zagreb	20.2.2016.	23	-	300	-
JavaCro'15	Rovinj	10.-12.5.2015.	46	5	200	11
Javantura v2	Zagreb	15.11.2014.	16	-	160	-
JavaCro'14	Poreč	11.-13.5.2014.	50	5	220	11
Javantura v1	Zagreb	22.2.2014.	12	-	150	-
WebCamp 2013						-
HrOUG 2013						12
JavaCro'13						-
HrOUG 2012						13
WebCamp 2012						-
Java 2012	Tuhelj	29.-30.5.2012.	34	7	170	-
HrOUG 2011	Rovinj	18.-22.10.2011.	12 (od 96)	1 (od 9)	460	11

This is our 19th conference!!! ☺

#Javantura #JavaCro #HrOUG #proud

And we are still there ☺



OK, but let's talk about Java! ☺

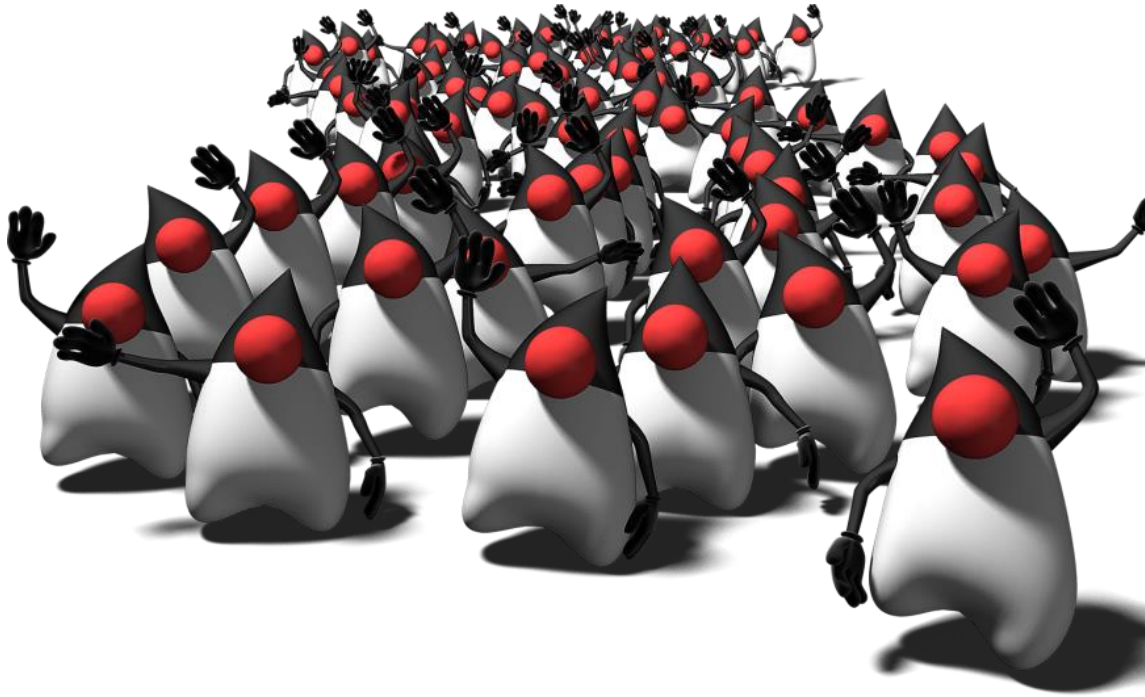
- *Before we start – some interesting facts:*
- Java does **not** stand for **Just Another Vague Acronym** ☺
- Invented almost **by accident**
  - They were building a new language for set-top box as a "cleaned up" C++ version
- Originally designed for **interactive TV** and **remote handheld devices**
  - Unfortunately, it was ahead of time
- Called **Oak** at the beginning
  - Later changed to Java because of **copyright** issues
- Java was named after a **coffee cup** slang word
  - Coffee imported from Indonesian **island** of Java (**Jawa**)

Some other interesting facts ☺

- **0xCAFEBABE** in class files is a tribute to the café
  - Where the **Green team** from **Project Stealth** went for coffee every day
- **James Gosling**, "the Father of Java", joined Amazon Web Services (AWS) team in 2017
  - After **Sun Microsystems**, Google (short), and Liquid Robotics
- **Joe Palrang**, the guy who created the Duke, also worked later on famous cartoon movies
  - The Simpsons, Shrek, Antz, Flushed Away, and Over the Hedge movies
- **JavaScript** got its name after Java
  - But only as a "marketing scam" just to make it more **popular**

So, what is… Java?

- **#1 Development Platform**
  - **Cloud**, **Microservices**, and **IoT**
- **Continued growth of Java** for **23 years**
- **A few Billion Devices** run Java
- **10 Million Java Developers** in the world
  - Many have **Java Certificates**
    - **OCA**, **OCP** & **OCM** for Java SE
    - **OCE** & **OCM** for Java EE
- **90%** of the **Fortune 500 companies** use **Java**
- But not only Java – **50+ JVM languages**
  - including Clojure, Groovy, Scala, JRuby, Jython, Fantom, Kotlin, Ceylon, Xtend, X10, LuaJ, Golo, Frege, Mirah, Eta… and JavaScript



Some Terminology

- **Java Platform, Standard Edition (Java SE)**
  - Specification of Java language, JVM, and core libraries
- **Java Development Kit (JDK)**
  - Tested (binary) implementation of Java SE
- **OpenJDK**
  - Open source reference implementation of Java SE
- **OpenJDK binary**
  - JDK built from OpenJDK source code

Current State of Java

- Are you still using Java **8**?
- Or you switched to Java **9** / **10**?
- Or the latest **Java 11**?
- What about Java **EE**?
- Well… let's explain

Java Platform today is:

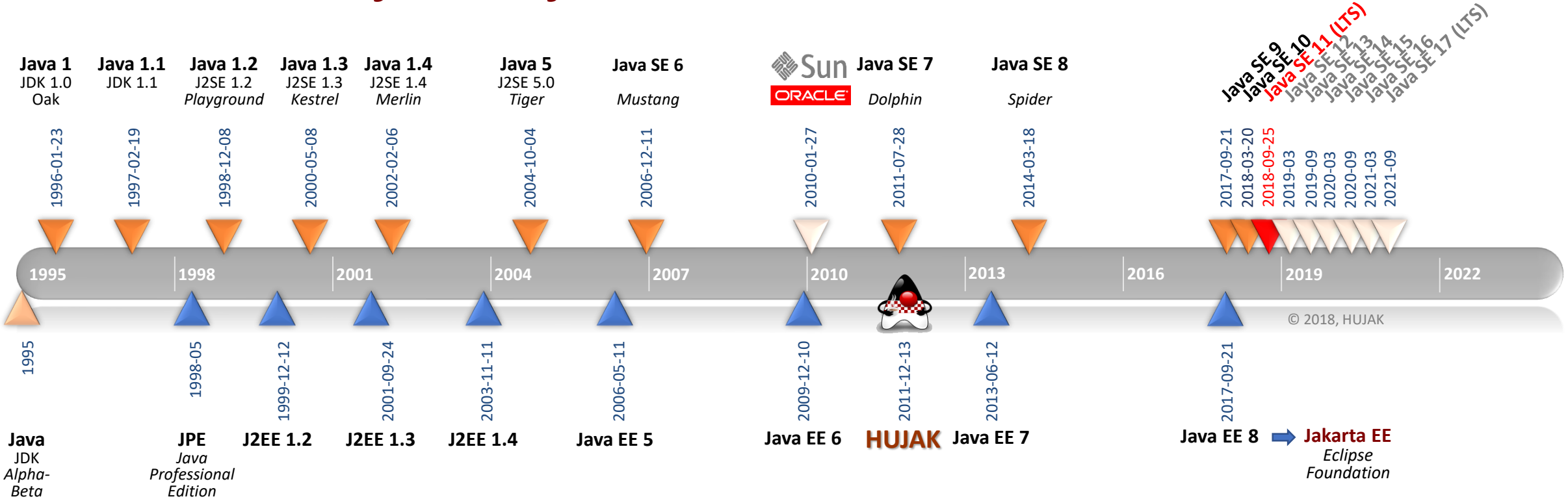
Stable **Secure** **Free** ?

However, commonly choose two out of three ☹

"Moving Java Forward Faster"

- *"For Java to remain competitive it must not just* ***continue to move forward*** *—* ***it must move forward faster****."*

Mark Reinhold



OpenJDK (New) Release Model

- **New Features** included (only) **when ready**
  - Not targeted for specific release, but released when feature **complete**
- **Feature release versions** released **twice a year**
  - Every **six months** in **March** and **September** (from JDK 9)
- **Update releases** will ship **quarterly**
  - in **January**, **April**, **July**, and **October**
- **Long-term support (LTS) feature release** every **three years**
  - Starting in September of 2018 with JDK 11
  - Updates will be available **for at least three years** and quite possibly longer
- **Time-Based Release Versioning** (JEP 322) openjdk.java.net/jeps/322
  - Revise the version-string scheme of the Java SE Platform and the JDK
  - Plans to name it by year and month (JEP 223), i.e. Java 18.3. – abandoned

JDK Version Numbering

- **$FEATURE.$INTERIM.$UPDATE.$EMERG**
  - **$FEATURE** is incremented every six months
    - Previously MAJOR
    - JDK **10** in March 2018, JDK **11** in September 2018, JDK **12** in March 2019…
  - **$INTERIM** is always **zero**, reserved for flexibility and future use
    - Previously MINOR
  - **$UPDATE** is incremented one month after $FEATURE is incremented, and every three months thereafter
    - Previously SECURITY
    - JDK **10.0.1** in April 2018, JDK **10.0.2** in July 2018, JDK **11.0.1** in October 2018…
  - **$PATCH** is emergency patch-release counter
    - Outside of planned schedule, incremented only when it's necessary to fix a critical issue

Available JDKs (and Licenses)

- **Oracle JDK** www.oracle.com/technetwork/java/javase/downloads/ **$$$?**
  - **Oracle Binary Code License (BCL)** with FoU (Field of Use) restrictions
- Many **OpenJDKs:**
- **Oracle OpenJDK** jdk.java.net/11/
  - **GNU General Public License version 2**, with the Classpath Exception (**GPLv2cpe**) with no restrictions
  - Security and bug fix updates every (and only for) **six months** (until next JDK, no overlap)
- **Azul's Zulu OpenJDK** www.azul.com/downloads/zulu/
  - Free, from JDK 6 to JDK 11, wide platform support
- **AdoptOpenJDK's OpenJDK** adoptopenjdk.net
  - Free, from JDK 8 to JDK 11, without commercial support, wide platform support
- **AdoptOpenJDK's OpenJDK based on OpenJ9** adoptopenjdk.net
  - OpenJ9 is former IBM commercial JVM, now open-sourced to Eclipse foundation
- **RedHat's OpenJDK**
- **SAP's SapMachine OpenJDK**
- **Other Linux distribution's OpenJDKs**

Demystifying "Free" Java

- **$free** as in **free beer** (cost) vs
  free as in **free speech** (what can you do)

- For **$free** use **OpenJDK binaries**
- For **free** use **OpenJDK** with **GPLv2+CE license**

- **Updates** refers to **code patches** – typically **$free**
- **Support** means **fixing bugs** and **answering questions** – was **never $free**

Oracle's JDK is (not) "free"

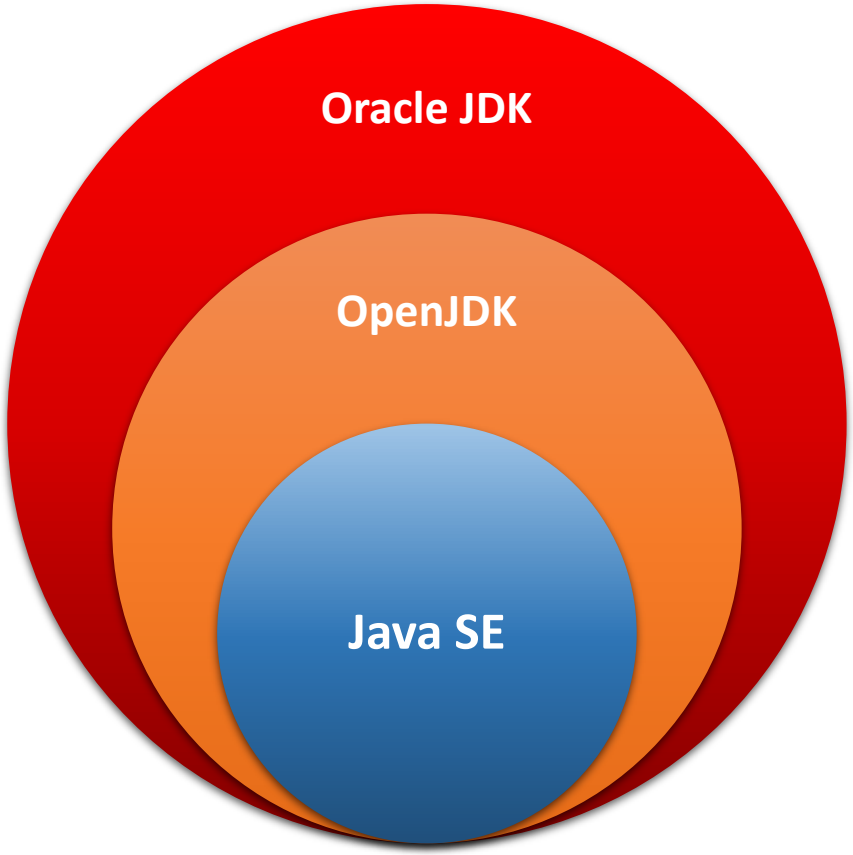
- LTS release **every 3 years** – **does not mean** 3 years of **free updates**
- **Oracle JDK 11** (and onward) can only be used in **production** with **commercial Java SE subscription**
  - Free JDK 11 (and later) are OpenJDK binaries
- **Oracle JDK 8** can be used **indefinitely for free**
  - **Without** any further **security patches** and **bug fixes**
- **Oracle** will lead and contribute to each new JDK (every 6 months)
  - For all JDK (Feature and LTE releases)
  - Will not backport updates, Java community need to do it for LTS release

Open Sourcing and Converged Binaries

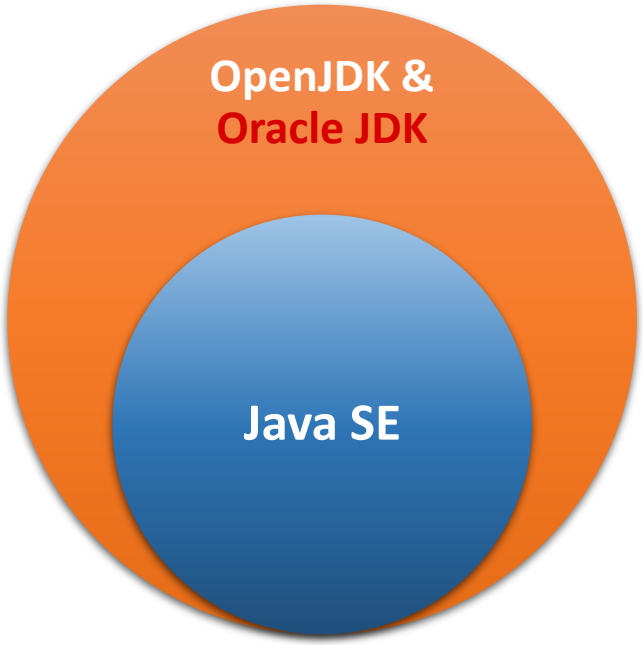
- Goal: **No functional difference** between OpenJDK and Oracle JDK in **JDK 11**
- **Open sourcing** closed-source parts of JDK
  - Flight recorder
  - Mission control
  - …
- **Removing** some closed-source parts
  - Browser Plugin
  - Java Web Start
  - JavaFX
- **Backwards Compatibility** – applications depending on Java SE should work

Converged Binaries

Up to JDK 10

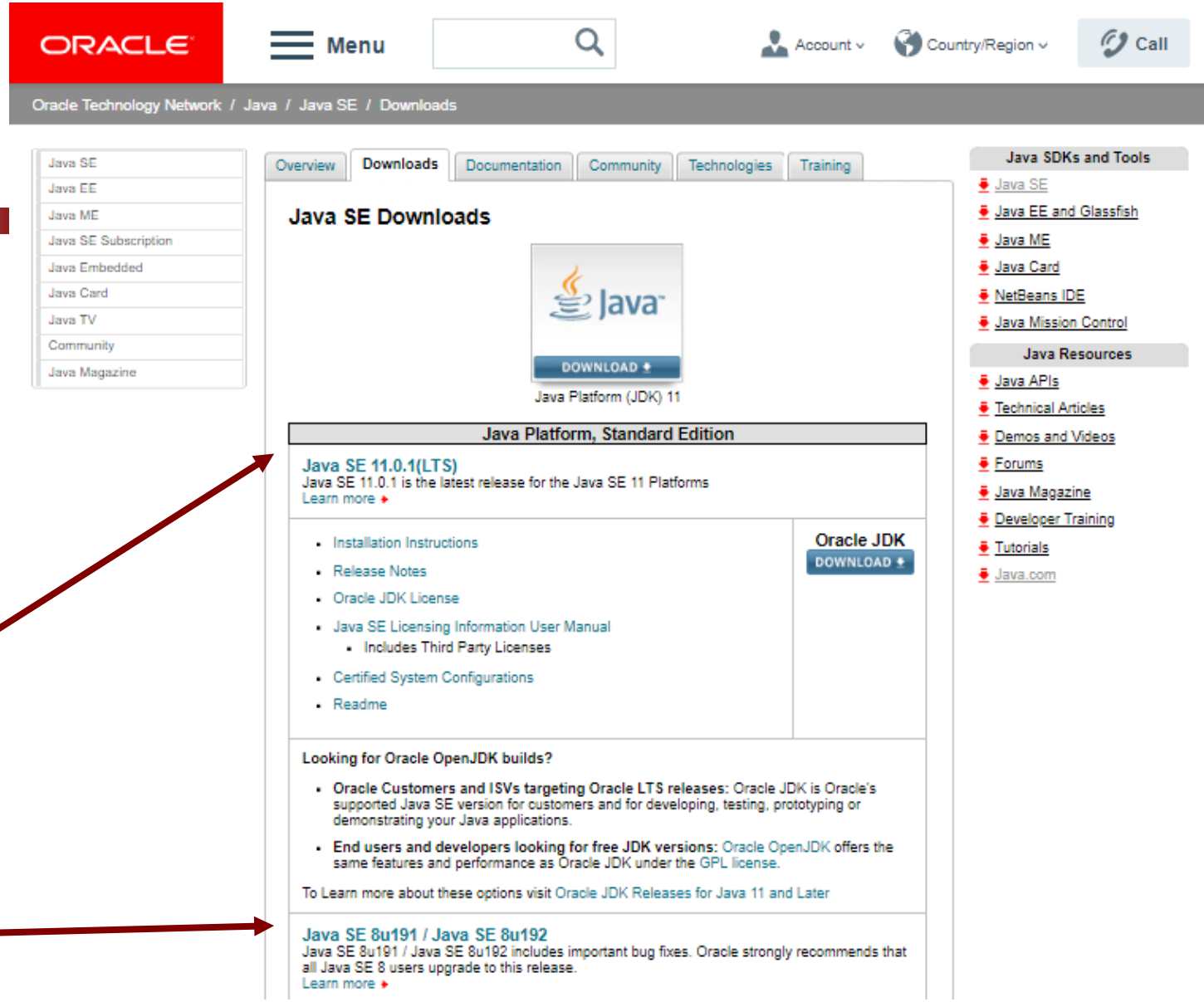


JDK 11 and later



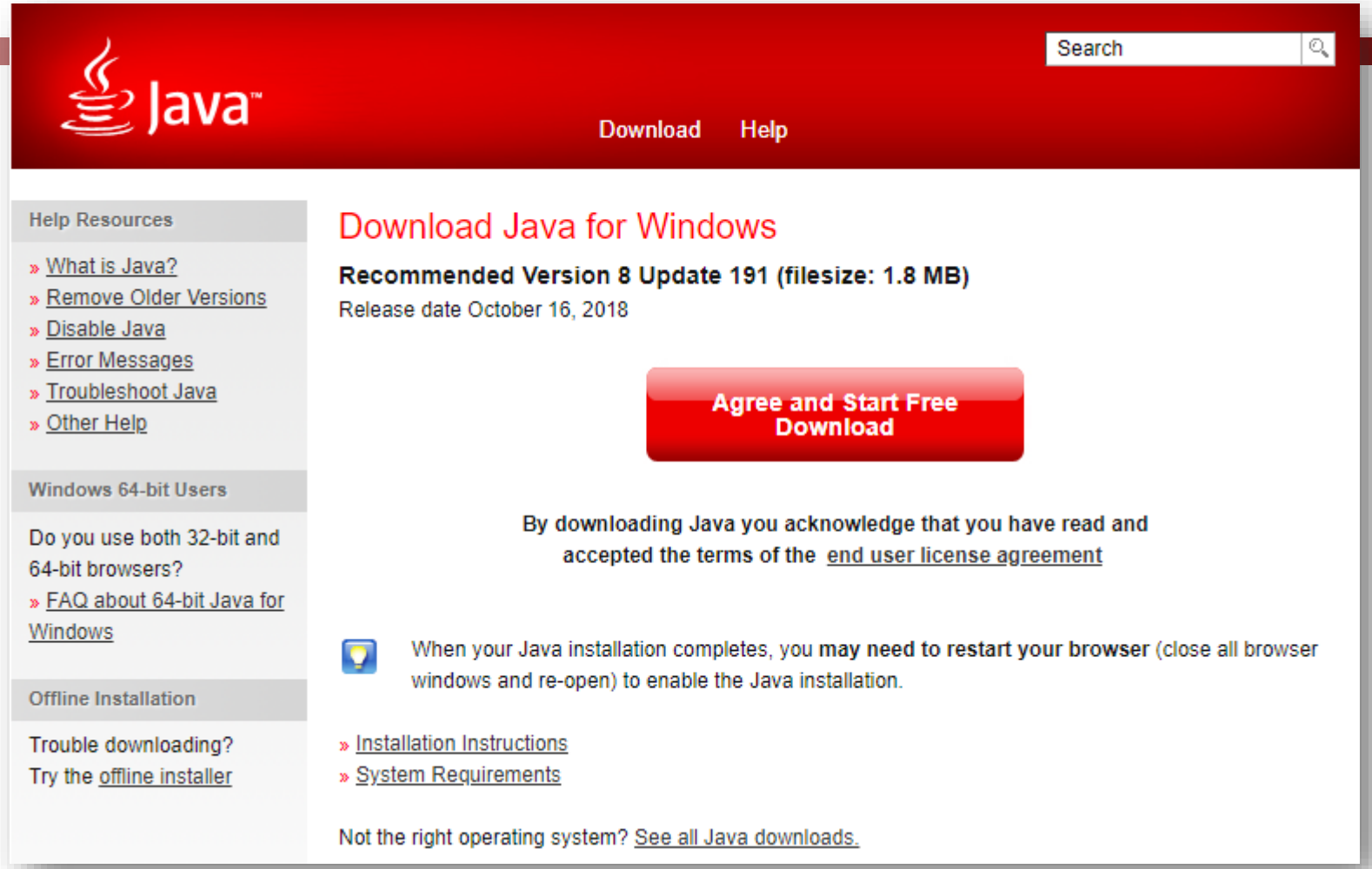
Java Download

- What/where to download **Java today**?
  - **OpenJDK**
  - Oracle **JDK**
  - Some other OpenJDK?
    - **Zulu** (Azul Systems) or **AdoptOpenJDK**
- Currently available downloads of Oracle's JDK:
  - Java SE **11.0.1**
  - Java SE **8u191**



What about www.java.com?

- Well…
  we don't know?!?



(Long Term) Support

- **Long Term Support (LTS)** for all releases is **not practical**
- **One Long Term Support release** every **three years**
  - Starting with **JDK 11** (September 2018), then **JDK 17** (September 2021), then **JDK 23**…
  - For Oracle's **commercial customers** updates available **for at least three years** or longer
- JDK **9** – supported until March 2018 (release of JDK 10)
- JDK **10** – supported until September 2018 (release of JDK 11)
- JDK **11** (September 2018) – supported until March 2019 (JDK 12)
- JDK **12** (March 2019) …
- JDK change every six months?

Public Updates and Support – from 7 till 17

Java SE Version	Public Release	Versions	Oracle Support	Commercial / Personal User End of Public Updates	Oracle's Premier / Extended Support
7	July 2011		Long Term Support (LTS)	-	July 2019 / July 2022
8	March 2014	8u191 – October 2018	Long Term Support (LTS)	January 2019 / December 2020	March 2022 / March 2025
9	September 2017	9.0.4+11 – January 2018	Short Term Support	March 2018	March 2018 / NA
10	March 2018	10.0.0 10.0.1 – April 2018 10.0.2 – July 2018	Short Term Support	September 2018	September 2018 / NA
11	**September 2018**	11.0.0 LTS **11.0.1 – October 2018** 11.0.2 – January 2019	**Long Term Support (LTS)**	*	September 2023 / September 2026
12	March 2019	-	Short Term Support	-	-
13	September 2019	-	Short Term Support	-	-
14	March 2020	-	Short Term Support	-	-
15	September 2020	-	Short Term Support	-	-
16	March 2021	-	Short Term Support	-	-
17	September 2021	-	Long Term Support (LTS)	-	-

* https://blogs.oracle.com/java-platform-group/oracle-jdk-releases-for-java-11-and-later

JDK 9/10 – old news?

- A lot of **significant changes**
- **Java Platform Module System (JPMS)**
  - All core Java libraries become modules (JEP 220)
  - 97 modules: 28 Java SE, 8 JavaFX, 59 JDK, 2 Oracle…
  - Most internal APIs encapsulated (JEP 260)
- **Deprecated APIs removed**
  - 1 package, many classes and methods
- **Redundant features eliminated**
  - Numerous deprecated GC options, jhat tool, TI hprof agent…
- Many **command line changes**
  - Removed 200+ -XX flags

JDK 10

- **JDK 10** was available since **March 20th, 2018**
  - JSR 383 – Oracle, IBM, Red Hat, SAP, Azul…
  - openjdk.java.net/projects/jdk/10/
- **109 new features** and **APIs**
- **JEPs** included
  - 286: Local-Variable Type Inference
  - 296: Consolidate the JDK Forest into a Single Repository
  - 304: Garbage-Collector Interface
  - 307: Parallel Full GC for G1
  - 310: Application Class-Data Sharing
  - 312: Thread-Local Handshakes
  - 313: Remove the Native-Header Generation Tool (javah)
  - 314: Additional Unicode Language-Tag Extensions
  - 316: Heap Allocation on Alternative Memory Devices
  - 317: Experimental Java-Based JIT Compiler
  - 319: Root Certificates
  - 322: Time-Based Release Versioning

Local Variable Type Inference

- **JEP 286** http://openjdk.java.net/jeps/286
- Extending **type inference** to declarations of **local variables** and **initializers**
  - Reducing the ceremony associated with writing Java
  - Maintaining the commitment to static type safety
- Examples:

```
var list = new ArrayList<String>();  // infers ArrayList<String>
var stream = list.stream();          // infers Stream<String>
var m = new HashMap <String, List<BigDecimal>>();
```

- Guidelines:
  - *Restricted to*: **local variables** with initializers, **indexes** in the enhanced for-loop, **locals** declared in a traditional for-loop
  - *Not available for*: **method** parameters, **constructor** parameters, **method return types**, **fields, catch formals** or any other kind of variable declaration
- *Don't blame language features for making developers write sh**y code* – Simon Maple

Garbage Collector Interface

- **JEP 304** http://openjdk.java.net/jeps/304
- Introducing a **clean GC interface** to improve source code isolation of different GCs
  - Better modularity for HotSpot internal GC code
  - Simpler to **add a new GC** to HotSpot without perturbing the current code base
  - Make it easier to **exclude a GC** from a JDK build
- Bits and pieces of GC source files scattered all over the HotSpot sources
  - Becomes an issue when implementing a new garbage collector
- BTW, some of our own experimenting with GCs (Parallel, CMS, G1…)
  - *Comparison of Garbage Collectors in Java Programming Language* at MIPRO 2018

Parallel Full GC for G1

- **JEP 307** http://openjdk.java.net/jeps/307
- **Improving G1 worst-case latencies** by making the **full GC parallel**
  - G1 GC (default GC since JDK 9) designed to avoid full collections
  - When concurrent collections can't reclaim memory fast enough – fall back full GC
  - Previous G1 implementation was using single threaded algorithm
    - Previous default was parallel collector (had parallel full GC)
- To minimize the impact for users experiencing full GCs, the G1 full GC was **made parallel** as well
  - Intends to parallelize the mark-sweep-compact algorithm
    - Use the same number of threads as the Young and Mixed collections do
  - Number of threads controlled by the *-XX:ParallelGCThreads* option
    - It will also affect the number of threads used for Young and Mixed collections

Handshakes, heap allocation, and Unicode

- **Thread-Local Handshakes**
  - **JEP 312** openjdk.java.net/jeps/312
  - Introduces how to execute a **callback on threads**, without performing a global VM safepoint
  - It is both possible and cheap to stop individual threads and not just all threads or none
- **Heap Allocation on Alternative Memory Devices**
  - **JEP 316** openjdk.java.net/jeps/316
  - Allocate Java object **heap** on an **alternative memory device** (e.g. NV-DIMM)
- **Additional Unicode Language-Tag Extensions**
  - **JEP 314** openjdk.java.net/jeps/314
  - Enhance **java.util.Locale** and related APIs to implement additional **Unicode extensions of language tag syntax** (BCP 47)

Some Housekeeping

- **Experimental Java-Based JIT Compiler (Graal)**
  - **JEP 317** openjdk.java.net/jeps/317
  - Performance ☺ – hotspots compiled to native
  - Enable **Graal** (Java-based JIT compiler) to be used as experimental JIT compiler

    ```
    -XX:+UnlockExperimentalVMOptions -XX:+UseJVMCICompiler
    ```
- **Root Certificates**
  - **JEP 319** openjdk.java.net/jeps/319
  - Provide a **default set of root Certification Authority (CA) certificates** in JDK
  - Open source the root certificates in Oracle's Java SE Root CA program
- **Consolidate the JDK Forest into a Single Repository**
  - **JEP 296** openjdk.java.net/jeps/296
  - Combine the various repositories of JDK forest into a **single repository**
  - Simplify and streamline development (FX not included)
- **Remove the Native-Header Generation Tool (*javah*)**
  - **JEP 313** openjdk.java.net/jeps/313
  - Remove the *javah* tool from the JDK, superseded by superior functionality in ***javac***

New APIs

- **73 additional new APIs**
  - **copyOf**(Collection) in List, Set and Map
  - Optional.**orElseThrow**() – get or throw
  - **toUnmodifiable**List/Map/Set
- Some APIs **removed**
  - Based on **Java SE 10 (18.3) (JSR 383) Proposed Final Draft Specification** cr.openjdk.java.net/~iris/se/10/pfd/java-se-10-pfd-spec-01/#APIs-removed
  - Optional annotation element `forRemoval=true` to previously deprecated API elements
  - Remove deprecated methods Runtime.getLocalized{Input,Output}Stream
  - Remove deprecated pre-1.2 SecurityManager methods and fields
  - De-deprecate XMLInputFactory.newFactory()

JDK 11

- **JDK 11** is in **General Availability**
  - JSR 384 – Oracle, IBM, Red Hat, SAP, Azul…
  - openjdk.java.net/projects/jdk/11/
- **90** new features in JDK 11
  - Post by Simon Ritter
  - https://www.azul.com/90-new-features-and-apis-in-jdk-11/
  - Less Developer **Visible** Features
- What is inside?

- **JEPs** included
  - 181: Nest-Based Access Control
  - 309: Dynamic Class-File Constants
  - 315: Improve Aarch64 Intrinsics
  - 318: Epsilon: A No-Op Garbage Collector
  - 320: Remove the Java EE and CORBA Modules
  - 321: HTTP Client (Standard)
  - 323: Local-Variable Syntax for Lambda Parameters
  - 324: Key Agreement with Curve25519 and Curve448
  - 327: Unicode 10
  - 328: Flight Recorder
  - 329: ChaCha20 and Poly1305 Cryptographic Algorithms
  - 330: Launch Single-File Source-Code Programs
  - 331: Low-Overhead Heap Profiling
  - 332: Transport Layer Security (TLS) 1.3
  - 333: ZGC: A Scalable Low-Latency Garbage Collector
  - 335: Deprecate the Nashorn JavaScript Engine
  - 336: Deprecate the Pack200 Tools and API

Local-Variable Syntax for Lambda Parameters

- **JEP 323** http://openjdk.java.net/jeps/323
- Extending **Local-Variable Type Inference** (JEP 286) – but now for **Lambda** expressions
  - Uniformity of local variables and lambdas
- Allow **var** when declaring **formal parameters** of **implicitly typed lambda expressions**

```
(var x, var y) -> x.process(y)
```

- For **all** formal parameters **or none** of them

```
(var x, y) -> x.process(y)      // Can't mix 'var' and 'no var' in implicitly typed lambdas
```

- Explicitly typed lambda expressions continue to use data types for all their formal parameters

```
(var x, int y) -> x.process(y)   // Can't mix 'var' and data types in explicitly typed lambdas
```

- Not compromising the brevity of the shorthand syntax

```
var x -> x.foo()                 // is not allowed
```

Local-Variable Syntax for Lambda Parameters – Example

- Example (by Simon Ritter):

```
list.stream()
    .map((var s) -> s.toLowerCase())
    .collect(Collectors.toList());
```

- Lambda expressions already have type inference so use of var is not necessary:

```
list.stream()
    .map(s -> s.toLowerCase())
    .collect(Collectors.toList());
```

- When **adding an annotation** to Lambda parameter you have to use a (explicit) type, and we can use **var** instead

```
list.stream()
    .map((@Notnull var s) -> s.toLowerCase())
    .collect(Collectors.toList());
```

- Also causes changes to the Java Language Specification (JLS) :
  - Description of the var special identifier, Lambda parameters, Runtime evaluation of Lambda expressions, and Lambda syntax

Launch Single-File Source-Code Programs

- **JEP 330** http://openjdk.java.net/jeps/330
- Run a program supplied as a **single file** of Java source code
  - Reduce the ‘ceremony’ of running trivial applications
  - Including usage from within a script by means of "shebang" files and related techniques
- Example:

```
java HelloWorld.java
```

- Parameters:
  - **After** the name of the source file are passed as parameters when executing application
  - **Before** the name of the source file are passed as parameters to launcher after code has been compiled
- Example:

```
java -classpath /home/foo/java Hello.java Bonjour
```

- is equivalent to:

```
javac -classpath /home/foo/java Hello.java
java -classpath /home/foo/java Hello Bonjour
```

Launch Single-File Source-Code Programs – Shebang files support

- Usage from within a script by means of "shebang" files and related techniques
- "Shebang" file – small utility single-file program starting with **#!**

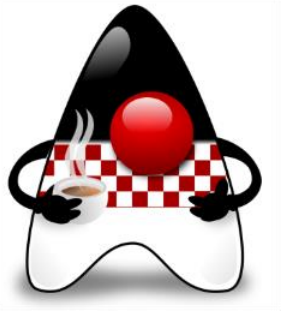
  `#!interpreter [optional-arg]`
  - On Unix-derived systems (Linux, macOS)
  - Allows a script or source code to be placed in any executable file (whose first line begins with #!), specifying name of a program to "execute" the contents of the file
- Reducing the need to even mention the Java launcher on the command line
- Simply included on the first line of the source file
- Example:

```
#!/usr/bin/java --source 11
    public class HelloWorld {
    ...
```

- However, necessary to specify the **`–source`** flag with the version of Java

HTTP Client (Standard)

- **JEP 321** http://openjdk.java.net/jeps/321
- New API in JDK 9 to provide support for the HTTP Client protocol (JEP 110) with **HTTP/2 support**
  - Since JDK 9 introduced the Java Platform Module System (JPMS), it was included as an incubator module
  - Later updated in JDK 10
- **HTTP Client API** is now part of the Java SE 11 standard
- New module and package **java.net.http**
- Main types:
  - **HttpClient, HttpRequest, HttpResponse, WebSocket**
- API can be used **synchronously** or **asynchronously**
  - Asynchronous mode makes use of **CompletableFutures** and **CompletionStages**



HTTP Client (Standard) – Some details

- While incubating in JDK 9 and JDK 10, implementation was almost completely rewritten
- The implementation is now completely **asynchronous**
  - Previous HTTP/1.1 implementation was blocking
- Provides **non-blocking request and response** semantics through CompletableFutures
  - Which can be chained to trigger dependent actions
- Back-pressure and flow-control of request and response bodies is provided for via the Platform's **reactive-streams** support in the **java.util.concurrent.Flow** API
- Use of the **RX Flow** concept has been pushed down into the implementation
  - Eliminated many of the original custom concepts needed to support HTTP/2
- The flow of data can now be more easily **traced**
  - From user-level request publishers and response subscribers down to the underlying socket
  - Significantly reduces the number of concepts and complexity in the code
  - Maximizes the possibility of **reuse** between HTTP/1.1 and HTTP/2

Remove Java EE and CORBA Modules

- **JEP 320** http://openjdk.java.net/jeps/320
- Remove **Java EE and CORBA** modules from Java SE Platform and JDK
  - Modules **deprecated** in Java SE 9 with intent to remove them in a future
- Java SE 6 included a full Web Services stack (originally developed for the Java EE Platform):
  - JAX-WS (Java API for XML-Based Web Services)
  - JAXB (Java Architecture for XML Binding)
  - JAF (the JavaBeans Activation Framework)
  - Common Annotations

Remove Java EE and CORBA Modules #2

- **JEP 320** http://openjdk.java.net/jeps/320 cont'd
- At the time of inclusion, versions in Java SE and Java EE were identical
  - Except one package in Common Annotations
- Over time, versions in Java EE evolved (difficulties for versions in Java SE):
  - Technologies gained features that were not relevant to Java SE (like Common Annotations package for data sources in a Java EE container)
  - Maintenance problematic due to having to sync the Java SE (in OpenJDK) with the Java EE versions (in upstream repositories)
  - Possible to obtain standalone versions of the technologies from the upstream projects and deploy them – unfortunately, it was not widely used in practice
- With JPMS we can divide the monolithic **rt.jar** file into multiple modules
  - Additionally possible to create a Java runtime only with modules you need – reduces size

Remove Java EE and CORBA Modules

- Java.se.ee meta-module includes **six modules** that are **no longer part** of JDK:
  - **corba**
  - **transaction**
  - **activation**
  - **xml.bind**
  - **xml.ws**
  - **xml.ws.annotation**
- If you still use APIs from these modules in your code, you supply them as a **separate module** or **library**
- It seems that the **java.xml** modules, which are part of the JAX-WS, SOAP-based web services support are the ones that are causing most problems

Flight Recorder

- **JEP 328** http://openjdk.java.net/jeps/328
- **Low-overhead data collection framework** for **troubleshooting** Java applications on JVM
- Prior to JDK 11 it was a **commercial** feature in Oracle JDK binary
- Oracle eliminated functional differences between Oracle JDK and OpenJDK and it was contributed to the OpenJDK
- Goals:
  - Provides APIs for producing and consuming data as events
  - Provides a buffer mechanism and a binary data format
  - Allows the configuration and filtering of events
  - Provides events for the OS, the HotSpot JVM, and the JDK libraries
- Two new modules : **jdk.jfr** and **jdk.management.jfr**

New API in JDK 11

- A lot of the **new APIs** in JDK 11
  - For a complete list of API changes, comparison by Gunnar Morling
    - https://gunnarmorling.github.io/jdk-api-diff/jdk10-jdk11-api-diff.html
- 6 new classes and 8 methods in **java.security** modules
  - Specific to the changes of JEP 324 and JEP 329
- New methods
  - In 8 slides

New Methods in JDK 11

- **New methods** – java.**io**:
- java.io.**ByteArrayOutputStream**
  - void **writeBytes**(byte[]) – write all the bytes of the parameter to the output stream
- java.io.**FileReader**
  - Two new constructors that allow a Charset to be specified
- java.io.**FileWriter**
  - Four new constructors that allow a Charset to be specified
- java.io.**InputStream**
  - io.InputStream **nullInputStream**() – returns an InputStream that reads no bytes
- java.io.**OutputStream**
  - io.OutputStream **nullOutputStream**() – like dev/null
- java.io.**Reader**
  - io.Reader **nullReader**()
- java.io.**Writer**
  - io.Writer **nullWriter**()

New Methods in JDK 11 #2

- **New methods** – java.**lang**:
- java.lang.**Character**
  - String **toString**(int) – overloaded form takes an int instead of a char as Unicode code point
- java.lang.**CharSequence**
  - int **compare**(CharSequence , CharSequence) – compares two CharSequence instances lexicographically
  - Negative, zero, or positive if 1st CharSequence is lexicographically less than, equal to, or greater than 2nd CharSequence, respectively
- java.lang.ref.**Reference**
  - lang.Object **clone**() – confusing, maybe for the future
- java.lang.**System** and java.lang.**Runtime**
  - No new methods, **runFinalizersOnExit**() method removed
- java.lang.**Thread**
  - No additional methods, **destroy**() and **stop**(Throwable) methods removed

New Methods in JDK 11 #3

- **New methods** – java.**lang**:
- java.lang.**String**
  - boolean **isBlank**() – returns true if string is empty or contains only white space codepoints
  - Stream **lines**() – returns a stream of lines extracted from this string, separated by line terminators
  - String **repeat**(int) – returns a string whose value is the concatenation of this string repeated count times
  - String **strip**() – returns a string whose value is this string, with all leading and trailing whitespace removed (different whitespace treatment than in trim())
  - String **stripLeading**() – returns a string whose value is this string, with all leading whitespace removed
  - String **stripTrainling**() – returns a string whose value is this string, with all trailing whitespace removed
- java.lang.**StringBuffer** and java.lang.**StringBuilder**
  - New **compareTo**() method that takes a StringBuffer/StringBuilder and returns an int (lexographical comparison same as for CharSequence)

New Methods in JDK 11 #4

- **New methods** – java.**nio**:
- java.nio.**ByteBuffer**, **CharBuffer**, **DoubleBuffer**, **FloatBuffer**, **LongBuffer**, **ShortBuffer**
  - **mismatch**() - finds and returns the relative index of the first mismatch between this buffer and a given buffer
- java.nio.channels.**SelectionKey**
  - int **interestOpsAnd**(int) – atomically sets this key's interest set to the bitwise intersection ("and") of the existing interest set and the given value
  - int **interestOpsOr**(int) – atomically sets this key's interest set to the bitwise union ("or") of the existing interest set and the given value
- java.nio.channels.**Selector**
  - int **select**(java.util.function.Consumer, long) – selects and performs an action on the keys whose corresponding channels are ready for I/O operations with timeout
  - int **select**(java.util.function.Consumer) – as above, except without the timeout
  - int **selectNow**(java.util.function.Consumer) – as above, except it is non-blocking

New Methods in JDK 11 #5

- **New methods** – java.**nio** (cont'd):
- java.nio.file.**Files**
  - String **readString**(Path): Reads all content from a file into a string, decoding from bytes to characters using the UTF-8 charset.
  - String **readString**(Path, Charset): As above, except decoding from bytes to characters using the specified Charset.
  - Path **writeString**(Path, CharSequence, java.nio.file. OpenOption[]:Write a CharSequence to a file. Characters are encoded into bytes using the UTF-8 charset.
  - Path **writeString**(Path, CharSequence, java.nio.file. Charset, OpenOption[]: As above, except Characters are encoded into bytes using the specified Charset.
- java.nio.file.**Path**
  - Path **of**(String, String[]): Returns a Path by converting a path string, or a sequence of strings that when joined form a path string.
  - Path **of**(net.URI): Returns a Path by converting a URI

New Methods in JDK 11 #6

- **New methods** – java.**util**:
- java.util.concurrent.**PriorityBlockingQueue** and java.util.**PriorityQueue**
  - void **forEach**(java.util.function.Consumer): Performs the given action for each element of the Iterable until all elements have been processed or the action throws an exception.
  - boolean **removeAll**(java.util.Collection): Removes all of this collection's elements that are also contained in the specified collection (optional operation).
  - boolean **removeIf**(java.util.function.Predicate): Removes all of the elements of this collection that satisfy the given predicate.
  - boolean **retainAll**(java.util.Collection): Retains only the elements in this collection that are contained in the specified collection (optional operation).
- java.util.concurrent.**TimeUnit**
  - long **convert**(java.time.Duration): Converts the given time duration to this unit.
- java.util.function.**Predicate**
  - Predicate **not**(Predicate). Returns a predicate that is the negation of the supplied predicate
  - *Example*: convert `lines.stream().filter(s -> !s.isBlank())` to `lines.stream().filter(Predicate.not(String::isBlank))` with static import `lines.stream().filter(not(String::isBlank))`

New Methods in JDK 11 #7

- **New methods** – java.**util**:
- java.util.**Optional**, **OptionalInt**, **OptionalDouble**, **OptionalLong**
  - boolean **isEmpty**():If a value is not present, it returns true, otherwise it is false.
- java.util.regex.**Pattern**
  - Predicate **asMatchPredicate**(): I think this could be a hidden gem in the new JDK 11 APIs. It creates a predicate that tests if this pattern matches a given input string.
- java.util.zip.**Deflater**
  - int **deflate**(ByteBuffer): Compresses the input data and fills the specified buffer with compressed data.
  - int **deflate**(ByteBuffer, int): Compresses the input data and fills the specified buffer with compressed data. Returns the actual number of bytes of data compressed.
  - void **setDictionary**(ByteBuffer): Sets the preset dictionary for compression to the bytes in the given buffer. This is an overloaded form of an existing method that can now accept a ByteBuffer, rather than a byte array.
  - void **setInput**(ByteBuffer): Sets input data for compression. Also an overloaded form of an existing method.
- java.util.zip.**Inflater**
  - int **inflate**(ByteBuffer): Uncompresses bytes into the specified buffer. Returns the actual number of bytes uncompressed.
  - void **setDictionary**(ByteBuffer): Sets the preset dictionary to the bytes in the given buffer. An overloaded form of an existing method.
  - void **setInput**(ByteBuffer): Sets input data for decompression. An overloaded form of an existing method.

New Methods in JDK 11 #8

- **New methods –** javax.**print**, javax.**swing**, and jdk.**jshell**:
- javax.print.attribute.standard.**DialogOwner**
  - This is a new class in JDK 11 and is an attribute class used to support requesting a print or page setup dialog be kept displayed on top of all windows or some specific window.
- javax.swing.**DefaultComboBoxModel**, **DefaultListModel**
  - void **addAll**(Collection): Adds all of the elements present in the collection.
  - void **addAll**(int, Collection): Adds all of the elements present in the collection, starting from the specified index.
- javax.swing.**ListSelectionModel**
  - int[] **getSelectedIndices**(): Returns an array of all of the selected indices in the selection model in increasing order.
  - int **getSelectedItemsCount**(): Returns the number of selected items.
- jdk.jshell.**EvalException**
  - jshell.JShellException **getCause**(): Returns the wrapped cause of the throwable in the executing client represented by this EvalException or null if the cause is non-existent or unknown.

Nest-Based Access Control

- **JEP 181** http://openjdk.java.net/jeps/181
- Java supports **nesting of classes** through **inner classes**
  - Logically, the inner class is part of the same code entity as the outer class
  - However, it is compiled as a separate class
  - Synthetic bridge method is created by the compiler to provide access to the private field of the outer class
- Introducing the concept of **nests**
  - Two members of the same nest (e.g., outer and inner class) are **nestmates**
  - **NestHost** and **NestMembers** attributes are defined for the class file format
  - Useful also for other languages compiled to bytecodes that support nested classes
- This feature introduces three new methods to **java.lang.Class**:
  - Class **getNestHost**()
  - Class[] **getNestMembers**()
  - boolean **isNestmateOf**(Class)
- Changes to JVMS in Access Control

Dynamic Class-File Constants

- **JEP 309** http://openjdk.java.net/jeps/309
- Extension of the class-file format to support a new constant-pool form **CONSTANT_Dynamic** (or "***condy***")
  - Idea of a dynamic constant seems to be an oxymoron
  - However, similar to a final value in Java
  - Like invokedynamic but for class-file constants
- **Constant-pool value** uses a **bootstrap method** to determine the value at **runtime**, not at compile-time (unlike the other constants)
  - Value is therefore "dynamic", but (since its value is only set once) it is also "constant"
- Simplifications primarily aimed at development of new JVM languages and compilers that generate bytecodes
- Introduces **java.lang.invoke.ConstantBootstraps** class with **9 new bootstrap methods** for dynamically computed constants
- Changes to the JVMS in usage of invokespecial bytecode and Constant Pool

Cryptographic-related changes

- **Key Agreement with Curve25519 and Curve448**
  - **JEP 324** http://openjdk.java.net/jeps/324
  - Replacing existing elliptic curve Diffie-Hellman (ECDH) scheme with **Curve25519** and **Curve448**
  - Key agreement scheme defined by **RFC-7748**
- **ChaCha20 and Poly1305 Cryptographic Algorithms**
  - **JEP 329** http://openjdk.java.net/jeps/329
  - Implementation of **ChaCha20** and **ChaCha20-Poly1305** ciphers as specified in RFC 7539, replacing the older insecure RC4 stream cipher

Garbage Collection

- **ZGC A Scalable, Low Latency Garbage Collector**
  - **JEP 333** http://openjdk.java.net/jeps/333
  - New experimental garbage collector designed for applications that require a **large (multi-gigabyte) heap and low-latency**
  - Uses a **single generation heap** and performs most of GC **concurrently** with the application
  - **Read-barrier** that intercepts each read to an object from the application and ensures that the reference returned is correct
  - Eliminates issue of being able to relocate objects concurrently while application threads are running
  - Region-based (like G1), NUMA aware and compacting
  - Not intended as a general-purpose collector

Garbage Collection #2

- **Epsilon: A No-Op Garbage Collector**
  - **JEP 318** http://openjdk.java.net/jeps/318 (by Red Hat)
  - Epsilon Garbage Collector handles memory allocation but does not implement any actual memory reclamation mechanism of space occupied by unreferenced objects
  - Designed to test and compare GC performance with and without GC
  - For very short-lived tasks (like serverless functions in the cloud) which do not exceed the memory allocated to the heap

Other JEPs in JDK 11

- **Unicode 10**
  - **JEP 327** http://openjdk.java.net/jeps/327
  - Support for Unicode 10.0 standard with 8,518 new symbols
  - Includes more Emojis, Bitcoin symbol, Nüshu character set, as well as Soyombo and Zanabazar Square
- **Improve Aarch64 Intrinsics**
  - **JEP 315** http://openjdk.java.net/jeps/315 (by Red Hat)
  - Take advantage of specialized instructions in **Arm64** instruction set
  - Improves performance of sin(), cos() and log() methods of the java.lang.Math class

Housekeeping continues

- **Removals** in JDK 11
  - **Applets, Browser Plugin, Web Start, Java FX**
- **Deprecate the Nashorn Scripting Engine**
  - **JEP 335** http://openjdk.java.net/jeps/335
  - Deprecates **Nashorn** introduced in JDK 8 as a replacement of Rhino Javascript engine
  - Suggests using **Graal VM** as replacement
    - How that will work has not been evaluated
  - Completely remove Nashorn with associated APIs and ***jjs*** tool in the future
- **Deprecate the Pack200 Tools and APIs**
  - **JEP 336** http://openjdk.java.net/jeps/336
  - With **JPMS** in JDK 9, **Pack200** a compression scheme for JARs is no longer used
  - Deprecates pack200 and unpack200 tools, and Pack200 API in java.util.jar, and may be removed in a future

Others

- **Low-overhead Heap Profiling**
  - **JEP 331** http://openjdk.java.net/jeps/331 (by Google)
  - Provides a way to get information about Java object heap allocations from the JVM that:
    - Is low-overhead enough to be enabled by default continuously
    - Is accessible via a well-defined, programmatic interface
    - Can sample all allocations
    - Can be defined in an implementation-independent way (i.e., not limited to a particular GC algorithm or VM implementation)
    - Can give information about both live and dead Java objects
- **Transport Layer Security (TLS) 1.3**
  - **JEP 332** http://openjdk.java.net/jeps/332
  - Implementation of TLS 1.3 (RFC 8446) which provides significant security and performance improvements over previous versions
  - Does not extend to Datagram Transport Layer Security (DTLS)

JDK 12

- **JDK 12** is currently in Early Draft Review
  - JSR 386 – usual suspects: Oracle, IBM, Red Hat, SAP, Azul…
  - openjdk.java.net/projects/jdk/12/
- Schedule:
  - 2018/05 Expert Group formation
  - 2018/07 Early Draft Review
  - 2018/10 - 2018/11 Public Review
  - 2019/01 - 2019/02 Proposed Final Draft
  - **2019/03** **Final Release**
- JEPs targeted to JDK 12, so far:
  - **Switch Expressions** (JEP 325)
  - **Raw String Literals** (JEP 326)

More Long-term Future

- Project **Amber** – incubator for smaller, productivity-oriented **language features** and **simplifying syntax**
  - Local variable type inference, local variable syntax for lambdas, **lambda leftovers**, **raw string literals**, **pattern matching**, **switch expressions**…
- Project **Valhalla** – incubator project for **advanced JVM and language feature** candidates
  - **Value types** and **specialized generics**
- Project **Panama** – to interconnect JVM and native code
  - **Foreign function interface** (**FFI**) replacement for JNI
- Project **Loom** – to reduce complexity in writing concurrent applications
  - **Fibres** (JVM-level threads) and **continuations**
- Project **Metropolis** – JVM re-written in Java, i.e. "***Java on Java***"
  - Using Graal experience, easier porting, performance to be explored (AOT compiler)

Project Amber

- Project **Amber** includes:
  - **Local variable type inference** (JEP 286) – delivered in JDK 10
  - **Local variable syntax for lambda parameters** (JEP 323) – delivered in JDK 11
  - **Switch Expressions** (JEP 325) – expressions in switch statements (lambdas) – planned for JDK 12
  - **Raw string literals** (JEP 326) – use of single backquote – planned for JDK 12
  - **Lambda leftovers** (JEP 302) – underscore for unused parameters – in progress
  - **Pattern matching** (JEP 305) – switch statement with case for different types of objects – in progress
  - **Enhanced Enums** (JEP 301) – generic enums with type parameters – currently on hold
- More at openjdk.java.net/projects/amber/

Raw string literals (JEP 326) – examples

```
Runtime.getRuntime().exec
("\"C:\\Program Files\\foo\" bar");
```

```
Runtime.getRuntime().exec
(`"C:\Program Files\foo" bar`);
```

```
System.out.println
("this".matches("\\w\\w\\w\\w"));
```

```
System.out.println
("this".matches(`\w\w\w\w`));
```

```
String html =
"<html>\n" +
"    <body>\n" +
"                <p>Hello
World.</p>\n" +
"    </body>\n" +
"</html>\n";
```

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

Pattern matching (JEP 305) – example

```
String formatted;
switch (obj) {
    case Integer i:
         formatted = String.format("int %d", i);
         break;
    case Long l:
         formatted = String.format("long %d", l);
         break;
    case Double d:
         formatted = String.format(“double %f", d);
         break;
    case String s:
         formatted = String.format("String %s", s);
         break
    default:
         formatted = obj.toString();
}
```

Switch Expressions (JEP 325) – example

```
int numLetters;
switch (day) {
    case MONDAY:
    case FRIDAY:
    case SUNDAY:
        numLetters = 6;
        break;
    case TUESDAY:
        numLetters = 7;
        break;
    case THURSDAY:
    case SATURDAY:
        numLetters = 8;
        break;
    case WEDNESDAY:
        numLetters = 9;
        break;
    default:
        throw new IllegalStateException("Hmm: " + day);
};
```

```
int numLetters = switch (day) {
    case MONDAY, FRIDAY, SUNDAY -> 6;
    case TUESDAY -> 7;
    case THURSDAY, SATURDAY -> 8;
    case WEDNESDAY -> 9;
    default -> throw new
       IllegalStateException(
       "Hmm: " + day);
};
```

Project Valhalla

- Incubator project for advanced Java VM and language feature candidates
- Problem:
  - Java uses **primitives** for performance and **objects** for OO, encapsulation, polymorphism, inheritance
  - But no **ArrayList<int>** ☹
  - If we use Integer than (un)boxing, creation of object, heap, indirection reference…
- **Value Objects (JEP 169)** – "codes like a class, works like a primitive"
  - Supports methods, fields, implements interface, encapsulation, generic…
  - Doesn't support mutation or sub-classes
- **Generics over Primitive Types (JEP 218)** – extends generic types to support the specialization of generic classes and interfaces over primitive types
- More at openjdk.java.net/projects/valhalla/

Project Panama and Project Loom

- Project **Panama** – interconnecting JVM and native code
  - Featuring **native function calling** from the JVM and **native data access** from the JVM
  - **Foreign function interface (FFI)** replacement for JNI
  - More at http://openjdk.java.net/projects/panama/
- Project **Loom** – reducing complexity in writing concurrent applications
  - Alternative, **user-mode thread implementations**, **delimited continuations**, and other constructs involving **call-stack manipulation**
  - Proposal for lightweight **fibres** (JVM-level threads) as alternative implementation of threads, managed by schedulers like `ForkJoinPool`, written in Java
  - Java programming model of ordinary Java threads would be preserved while performance is improved and the footprint reduced
  - Less memory and almost zero overhead when task switching
  - More at http://openjdk.java.net/projects/loom/

Backwards Compatibility

- It will be **respected** ☺
- But with **no guarantees** ☹
- New version may include **breaking changes**
- Anything for removal will be **deprecated first**
  - Minimum of **one release warning** (6+ months)

Container Awareness and ~~Java~~ Jakarta EE

- JVM more **Docker container aware**
  - Uses container CPU count and memory size

- **Open sourcing Java EE**
  - **Jakarta EE** as a part of **Eclipse Foundation**
  - jakarta.ee/ – *The New Home of Cloud Native Java*
  - **Jakarta EE Developer Survey 2018**, Eclipse Foundation
    - jakarta.ee/news/2018/04/24/jakarta-ee-community-survey/
  - Current status
    - blogs.eclipse.org/post/mike-milinkovich/jakarta-ee-status-%E2%80%93-september-2018-update

Is Java really "Moving Forward Faster"?

- Well… yeah ☺
- **Much more frequent Java releases**
- **Faster access to new features**
- **Many new improvement ideas**
- **A lot of maintenance and housekeeping**
- **Java remains free**

- BTW, what about JavaOne?

No more "JavaOne"?

- In 2018 JavaOne is **larger than ever**
- However, it goes by a **new name**… ☹
- **Oracle Code One –**
  a conference for all developers
  - October 22-25, San Francisco
  - Usual prices $1400-$2000 ☹
- Keynotes:
  - www.oracle.com/code-one/keynotes.html
- On Demand Streaming:
  - www.oracle.com/code-one/on-demand.html

- **11 tracks**:
  - Core Java Platform
  - Java Server-Side Development and Microservices
  - Java Ecosystem
  - Containers, Serverless, and Cloud
  - Emerging Technologies
  - Modern Web
  - Development Tools
  - DevOps and Pipelines
  - Developer Community
  - Database, Big Data, and Data Science
  - MySQL

Back to Croatian Reality

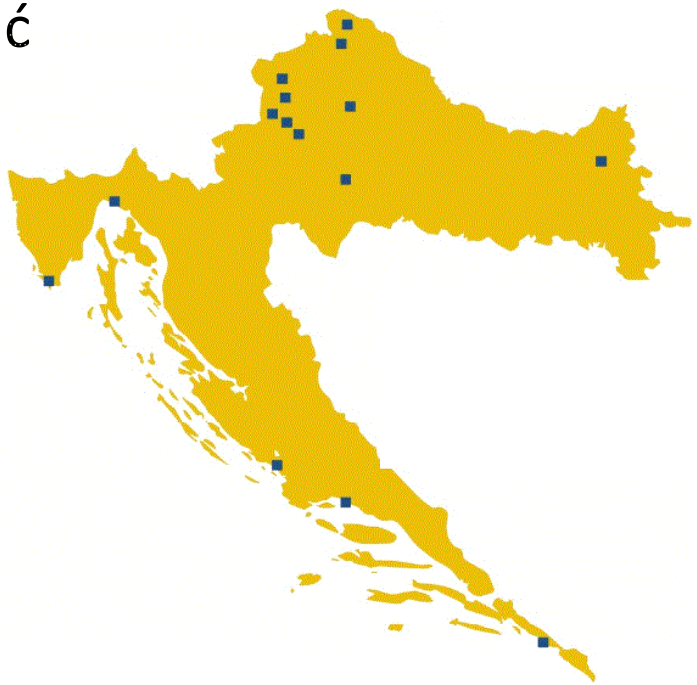
- But before that – one nice link: https://snyk.io/blog/jvm-ecosystem-report-2018

A few nice things in 2017/2018…

- **Java Zagreb meetups** – many great meetups so far
- **Java in high schools initiative**
  - Together with **Oracle Academy**
- **Croatian Makers league** continues
  - Micro:bit, Logo, mBot, Scratch, Arduino, Little Bits…
- Program **Digitalna akademija**
  - ScratchJr, RunMarco, Studio Code, Play Lab, Scratch i App studio, micro:bit, Arduino…
- **Code Club Croatia** and udruga **Programerko**
- Udruga za darovitu djecu "**Dar**"
- Great **Javantura** and **JavaCro** conferences

Where to study IT in Croatia?

- **Java** at **Universities**
  - Java is **#1** for decades!!! ☺
- Where to study **computing / computer science / information technology**?
  - **15+ cities**: Čakovec, Dubrovnik, Krapina, Križevci, Osijek, Pula, Rijeka, Sisak, Split, Šibenik, Varaždin, Velika Gorica, Zabok, Zagreb, Zaprešić
  - **33+ educational organizations**, including:
    - 6 public universities
    - 13 private high schools
  - **80+ educational programs**
    - Undergraduate (3-4 years)
    - Graduate (1-2 years)
    - Professional
    - Postgraduate



Javantura conference

- One **Saturday** in February
- **26 sessions**
- **300 attendees**
- Tickets with **50% discount for students**

Looking forward to
Javantura v6
on **February 23rd, 2019**
in **Zagreb** ☺

www.hujak.hr

Conferences HUJAK supports

droidcon

DORS/CLUC

QED

23. hroug
godišnja konferencija

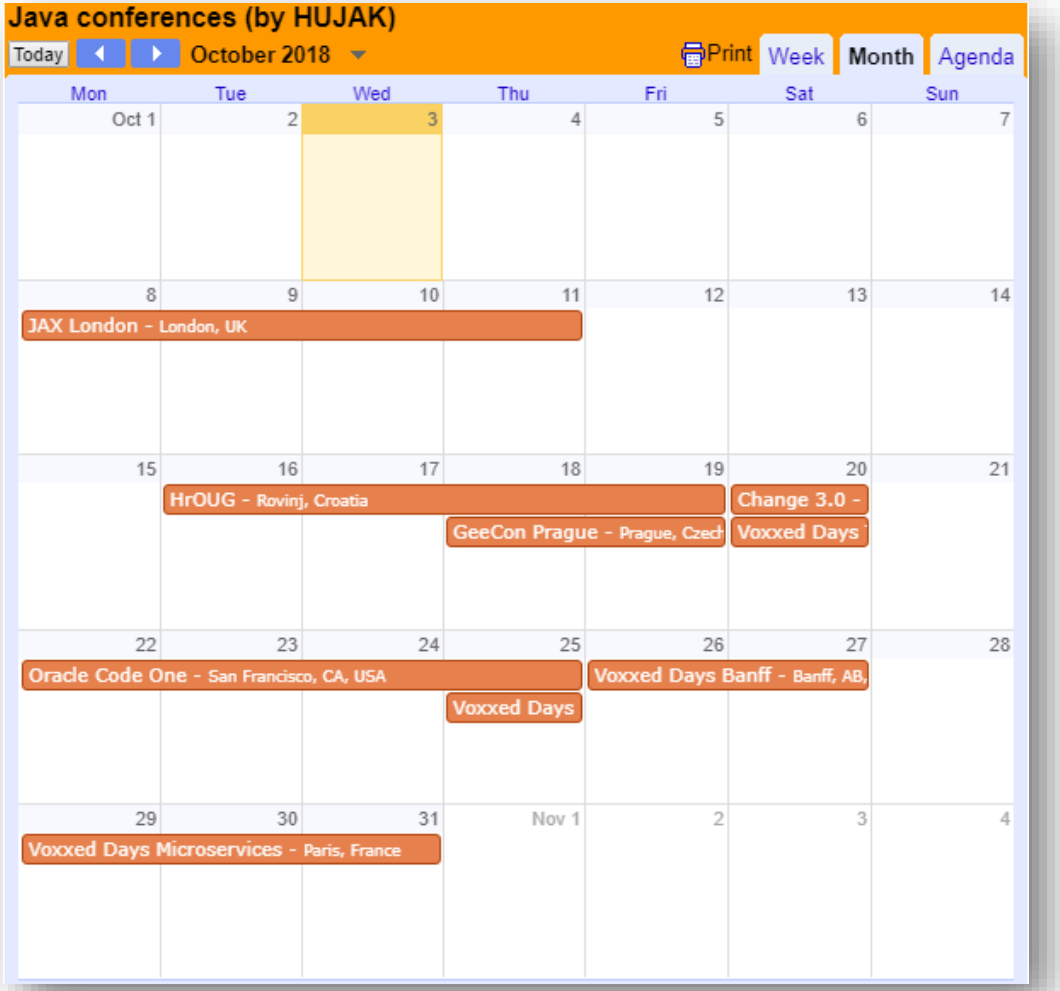
Shift
DEVELOPER CONFERENCE

change
powered by KING ICT

INFOBIP DEV DAYS

Calendar of Java-relates Conferences in EU

- Available at: hujak.hr/kalendar/
- If we are missing some please send email to info (at) hujak.hr
- Another great conference list at www.baeldung.com/java-conferences-europe



45+ company members

AGOA
ALTIMA BUSINESS INTEGRATOR
APIS IT
asseco SOUTH EASTERN EUROPE
CROZ
DABAR INFORMATIKA
AMPHINICY TECHNOLOGIES Nothing is Impossible.
dynniq
ecx.io an IBM Company
EVOLVA
Fina
infoart razvoj i održavanje softvarea
CORVUS
infobip
INOVATRENDI Inovativni trendovi d.o.o.
ISTARSKA KREDITNA BANKA UMAG d.d.
instantor
KING ICT
Hrvatski Telekom
LIFERAY
MPI
ITerago
etna
OptimIT
Pardus
PERSOLVO
identalia
NEOS
realnetworks
Sedam IT Primjena informatičkih tehnologija
SV GROUP
ERICSSON Ericsson Nikola Tesla d.d.
Steatoda
TRILIX
MENTAT LABS
SERENGETI CUSTOM SOFTWARE SOLUTIONS
S2 INFORMACIJSKI SUSTAVI d.o.o.
LibusoftCicom
BTB beta tau beta
srce
F
COMBIS
oradian

Thank you!

Partners & Friends

hroug
hrvatska udruga oracle korisnika

CARNet
HRVATSKA AKADEMSKA I ISTRAŽIVAČKA MREŽA

HROPEN

ORACLE ACADEMY

Agile Croatia

JUG.ba

HULK
hrvatska udruga linux korisnika

javasvet

openBLEND

BULGARIAN Java User Group

NetBeans User Group Serbia

kompetencijski centar za programsko inženjerstvo u otvorenim sustavima
Competence center for software engineering in open systems

SIOUG

JUGMK

časopis za IT profesionalce
MREŽA

NETOKRACIJA

R·I·T Croatia
Celebrating 20 years in Croatia

SorA
Škola otvorenog računarstva

Thank you & greetings from HUJAK!

- Web page **hujak.hr**
  - www.hujak.hr

- LinkedIn group **HUJAK**
  - www.linkedin.com/groups?gid=4320174

- Facebook group page **HUJAK.hr**
  - www.facebook.com/HUJAK.hr

- Twitter profile **@HUJAK_hr**
  - twitter.com/HUJAK_hr