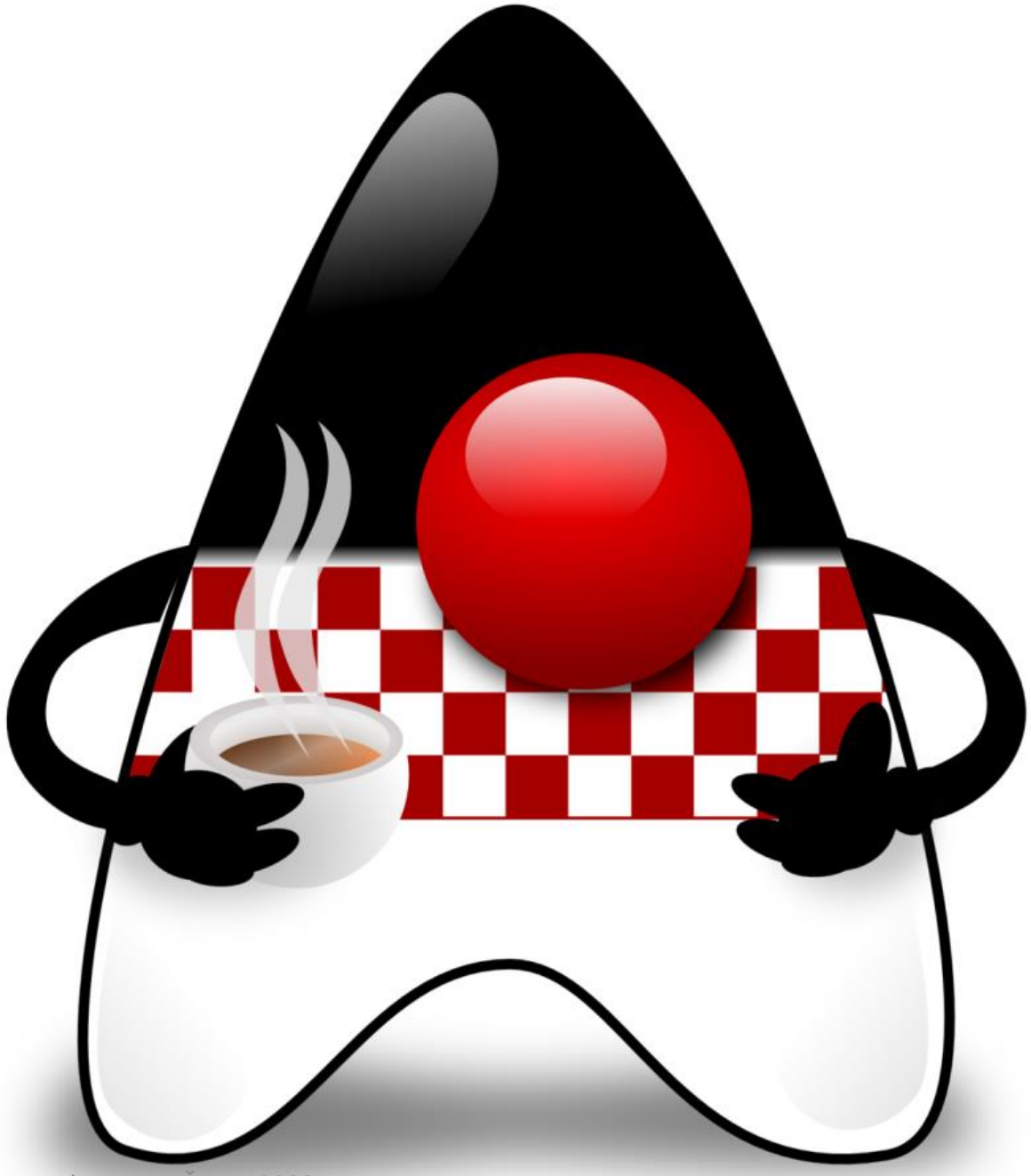
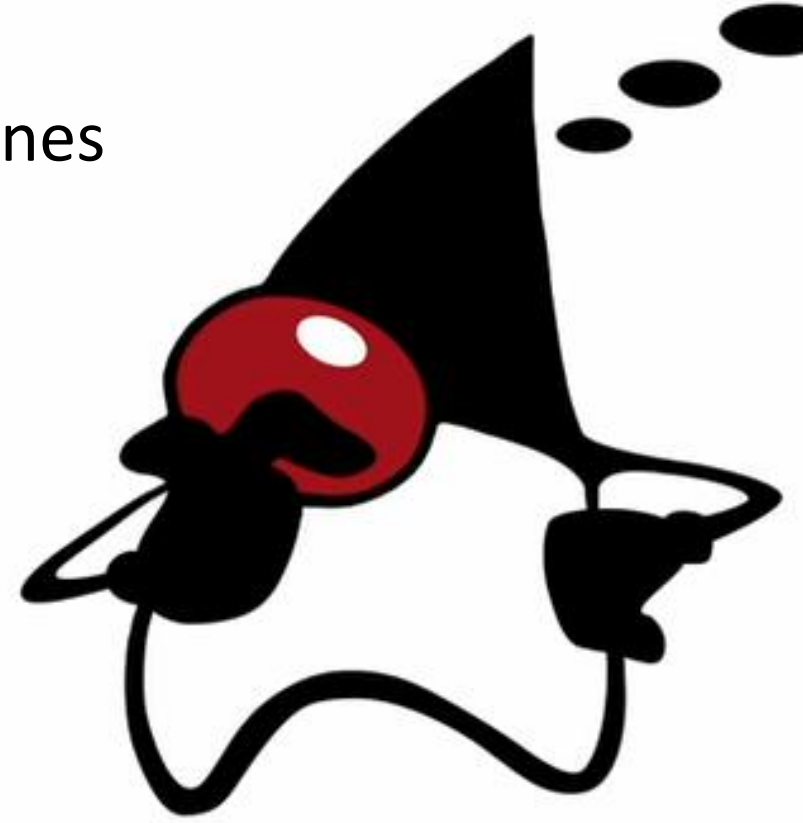
HUJAK
Community Keynote
From
Great Code and Features
to
Even Better Community

dr. sc. **Branko Mihaljević**
dr. sc. **Aleksander Radovan**
Stjepan Matijašević
dr. sc. **Martin Žagar**
HUJAK



Assessing the New Development Landscape

- New programming **languages**, **polyglotism & interoperability**
- New software development **paradigms**
- New **frameworks** or new (different) versions of old ones
- Modern application **solutions**
- Variety of **deployment models**
- **Cloud**-everywhere
- **Micro**services
- Anything/everything-**as-a-service**
- **…**

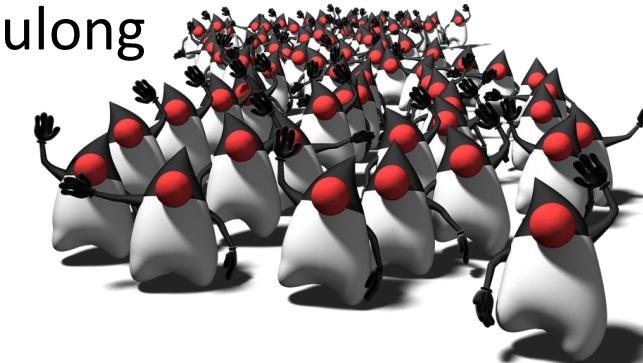


Java Facts

- **#1 Development Platform** (still!)
  - **Continued growth** for **27+ years**
- **#1 Programming Language**
  - In overall software development
- **60 Billion Active JVMs**
  - Expected to **grow** at over **9%** per year
- **38 Billion Cloud-based JVMs**
- **69%** of **Software Developers** run (some kind of…) Java apps

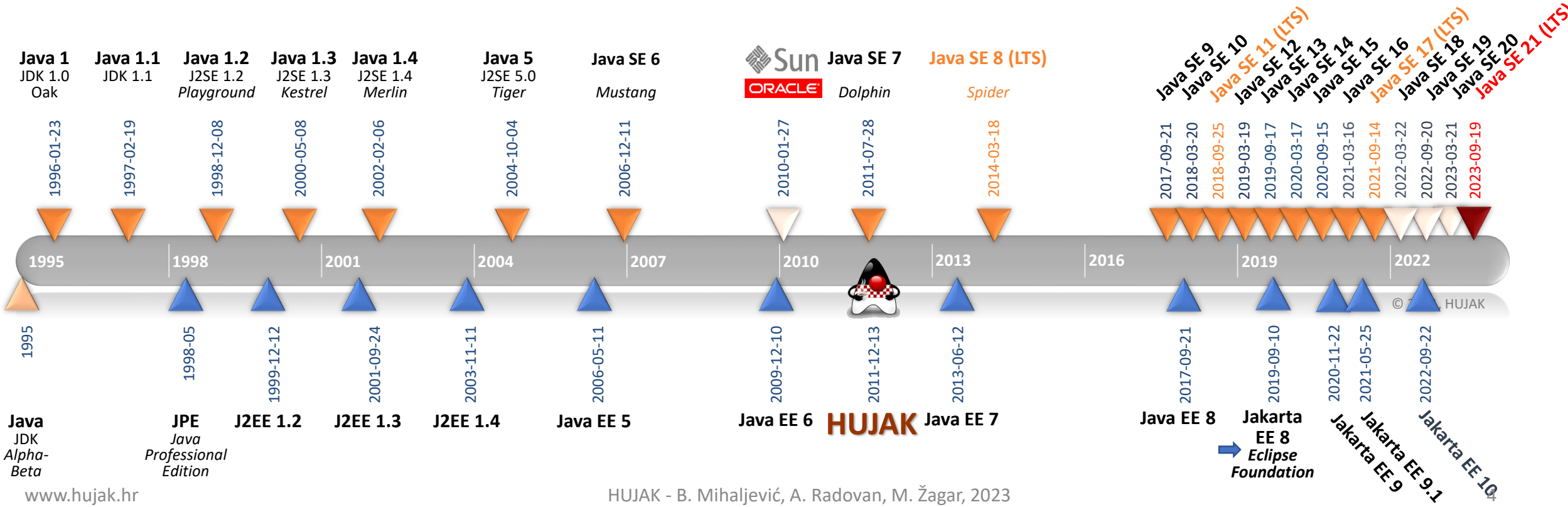
- **10 Million Java Developers**
  - With many **Java Certificates**

- **50+ JVM languages**
  - JVM languages: Groovy, Kotlin, Scala, Clojure, JRuby, Jython, Fantom, Ceylon, Xtend, X10, LuaJ, Golo, Frege, Mirah, Eta, JavaScript…
- And **other languages** with **GraalVM**
  - With Truffle and Sulong



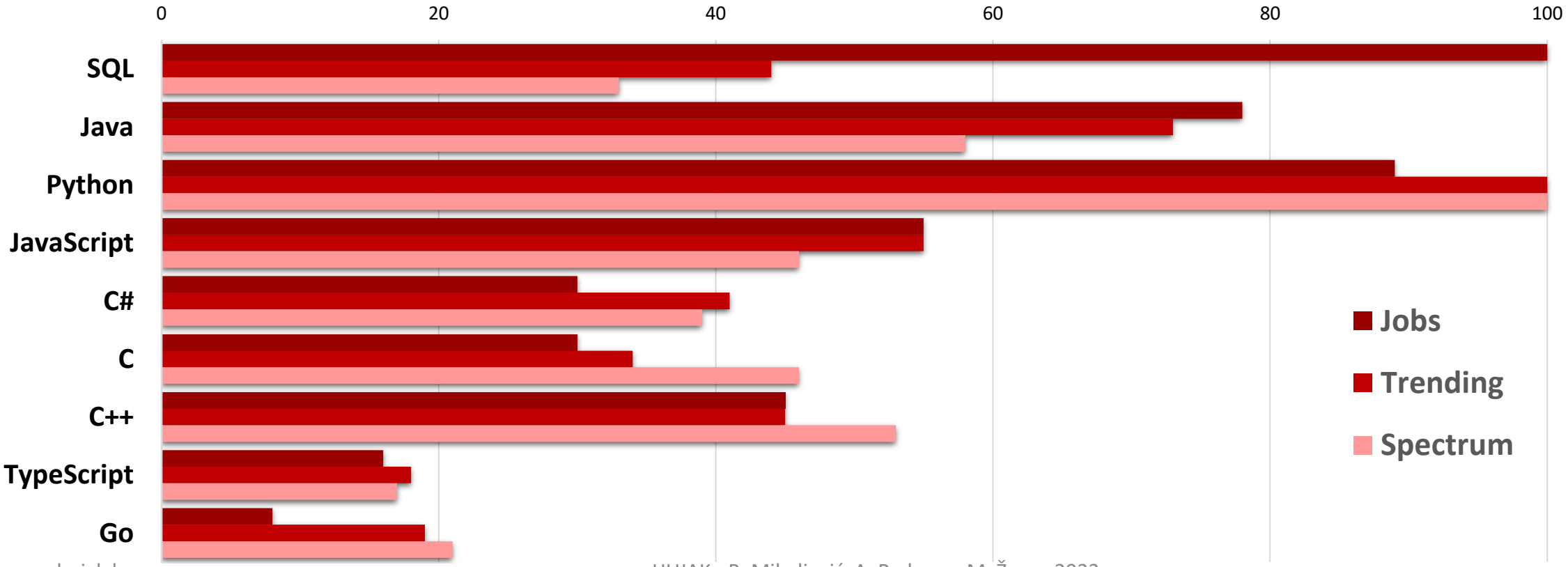
Java Timeline

- **27+ years** of history… a legend



Is Java still popular?

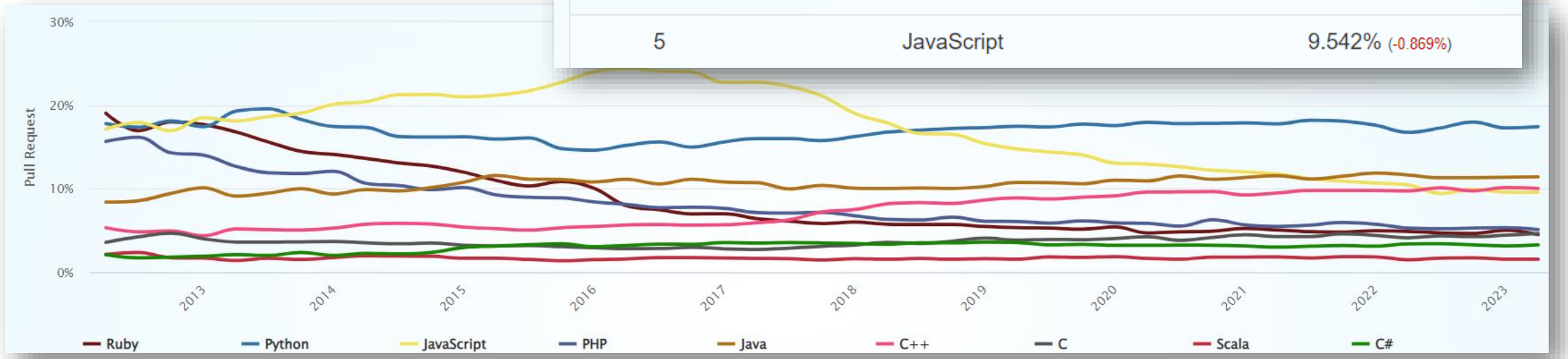
- **Top Programming Languages 2023** by IEEE Spectrum
  - Combining Google Search, Twitter, Stack Overflow, Reddit, IEEE Xplore, IEEE Jobs, CareerBuilder, GitHub



Is Java still popular?

- **GitHut 2.0 in Q2 2023**
  - **Pull Requests**

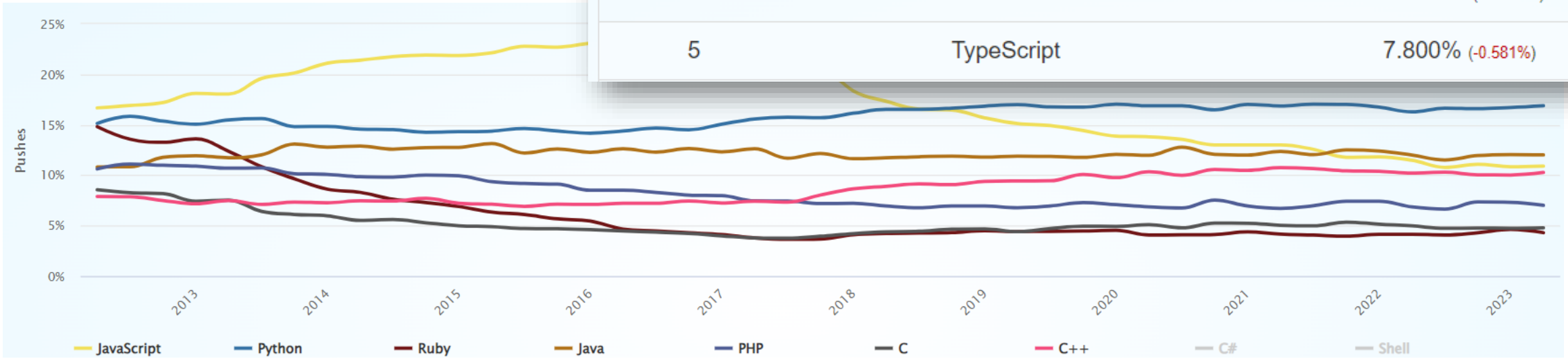
# Ranking	Programming Language	Percentage (YoY Change)
1	Python	17.355% (+0.673%)
2	Java	11.387% (-0.212%)
3	Go	10.877% (+1.574%)
4	C++	9.994% (+0.301%)
5	JavaScript	9.542% (-0.869%)



Is Java still popular?

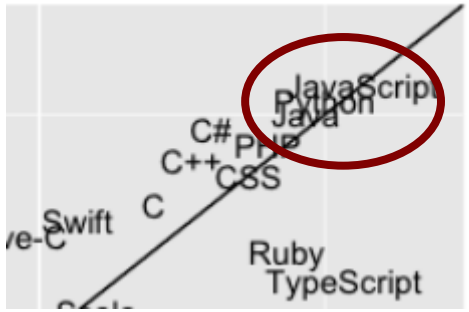
- **GitHut 2.0 in Q2 2023**
  - **Pushes**

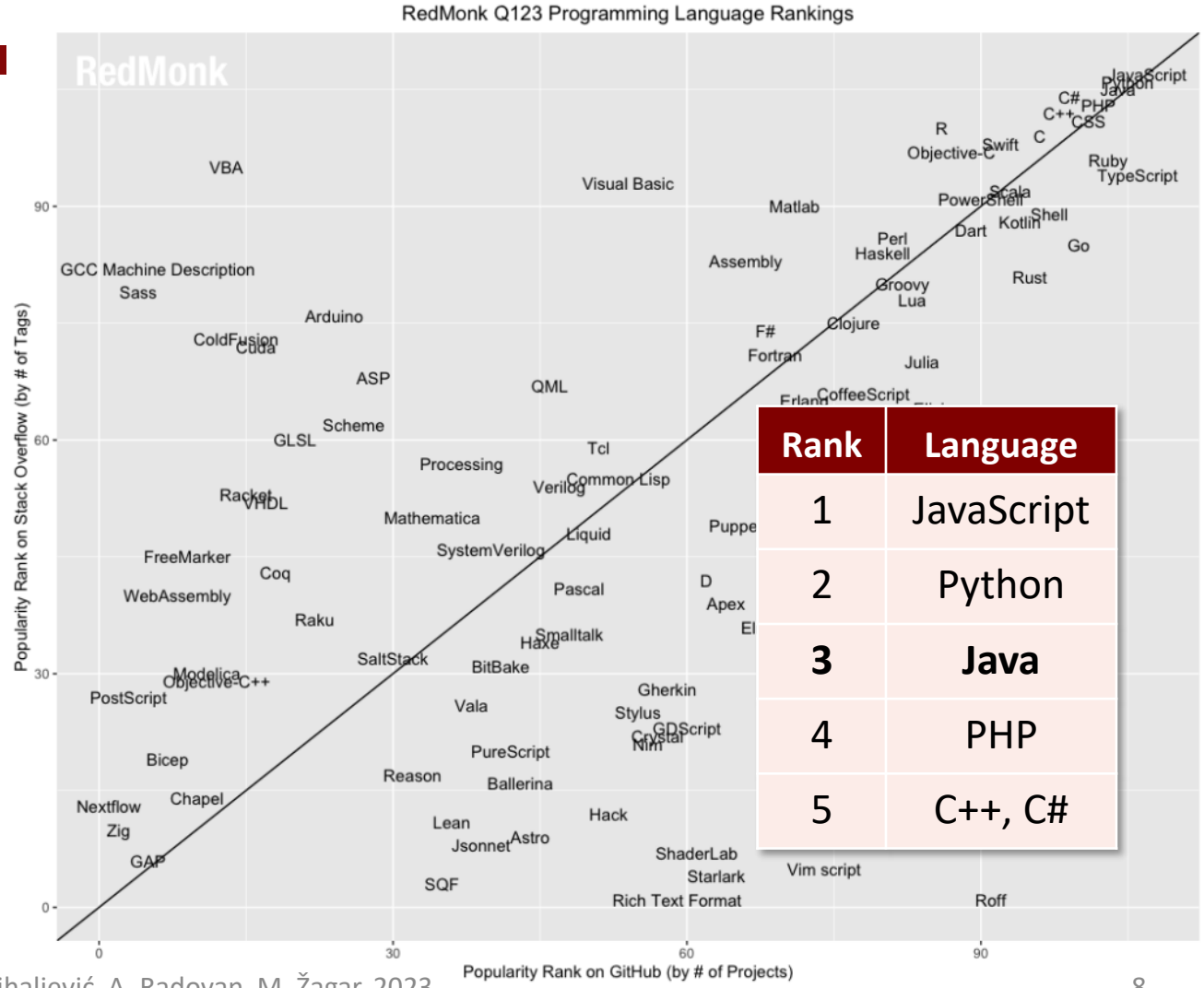
# Ranking	Programming Language	Percentage (YoY Change)
1	Python	16.758% (+0.594%)
2	Java	11.928% (+0.021%)
3	JavaScript	10.827% (-0.570%)
4	C++	10.195% (+0.055%)
5	TypeScript	7.800% (-0.581%)



Is Java still popular?

- **RedMonk Programming Language Rankings**: January 2023
  - Extraction of language rankings from **GitHub** and **Stack Overflow**
  - Combining them to reflect both code (GitHub) and discussion (Stack Overflow) traction





Rank	Language
1	JavaScript
2	Python
3	**Java**
4	PHP
5	C++, C#

Available JDKs?

- One of many OpenJDKs or **Oracle JDK**

- **Oracle OpenJDK**
- **Amazon's Corretto OpenJDK**
- **Adoptium / Eclipse Temurin OpenJDK**
- **Azul Zulu OpenJDK**
- **RedHat's OpenJDK**
- **IBM Semeru OpenJDK**
- **Linux distribution's OpenJDK**
- **Alibaba Dragonwell OpenJDK**
- **Bellsoft Liberica OpenJDK**
- **SAP SapMachine OpenJDK**
- **Microsoft OpenJDK**
- **...**
- **Oracle GraalVM CE or EE**

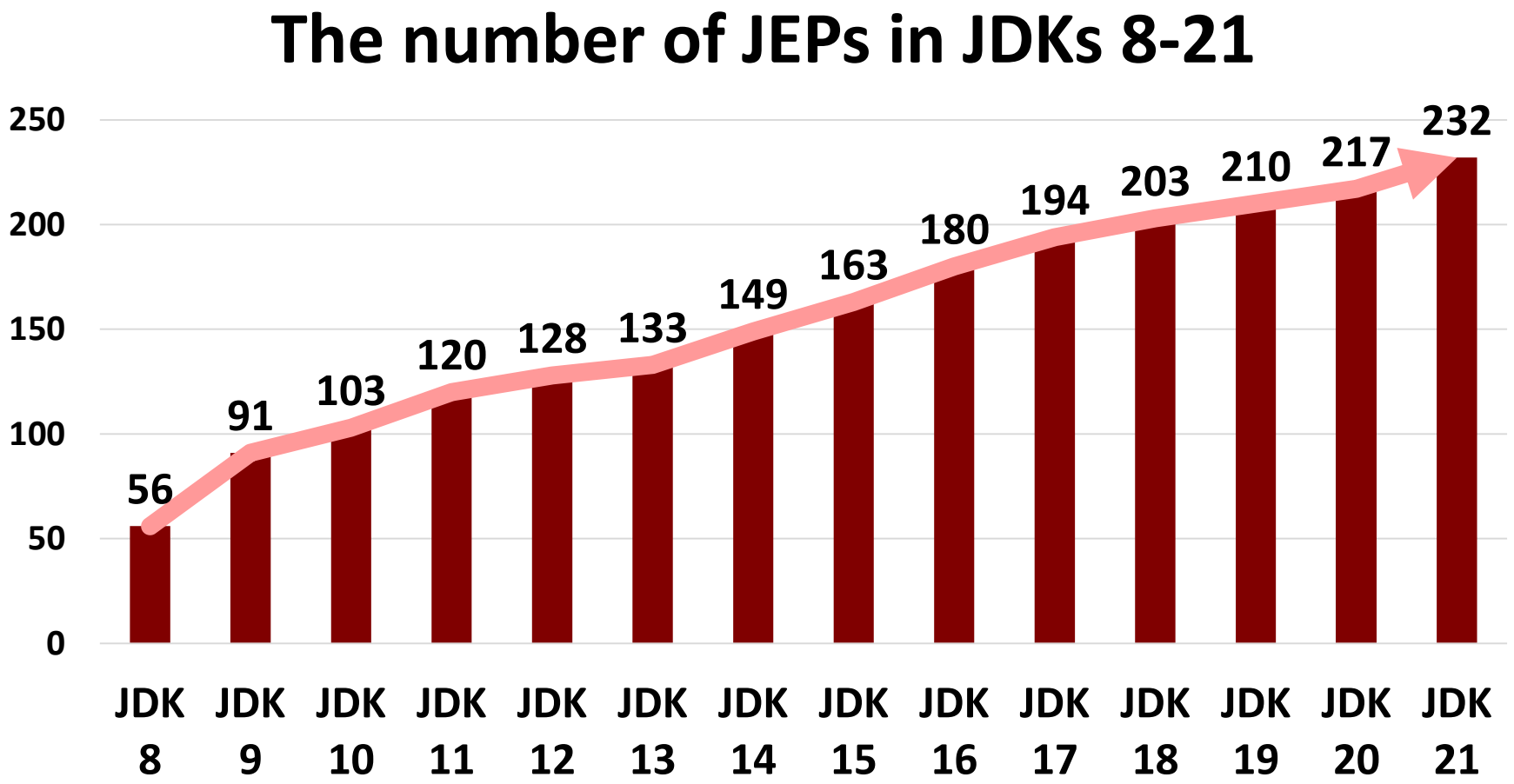
Is Java Moving Forward?

a) Variety of **Tools, Libraries** and **Frameworks**
   - **Stable evolution** - incrementally and predictably
   - Careful **backward compatibility**

b) Community **Trust, Acceptance**, and **Familiarity**
   - **No surprises** - open and transparent development model
   - **Community involvement** – entire community contributes to new features

c) Continuous **Innovation** and **Predictability**
   - **Innovative improvements** - respecting contemporary software development
   - **Cautious innovation** - gradually introducing language/platform enhancements

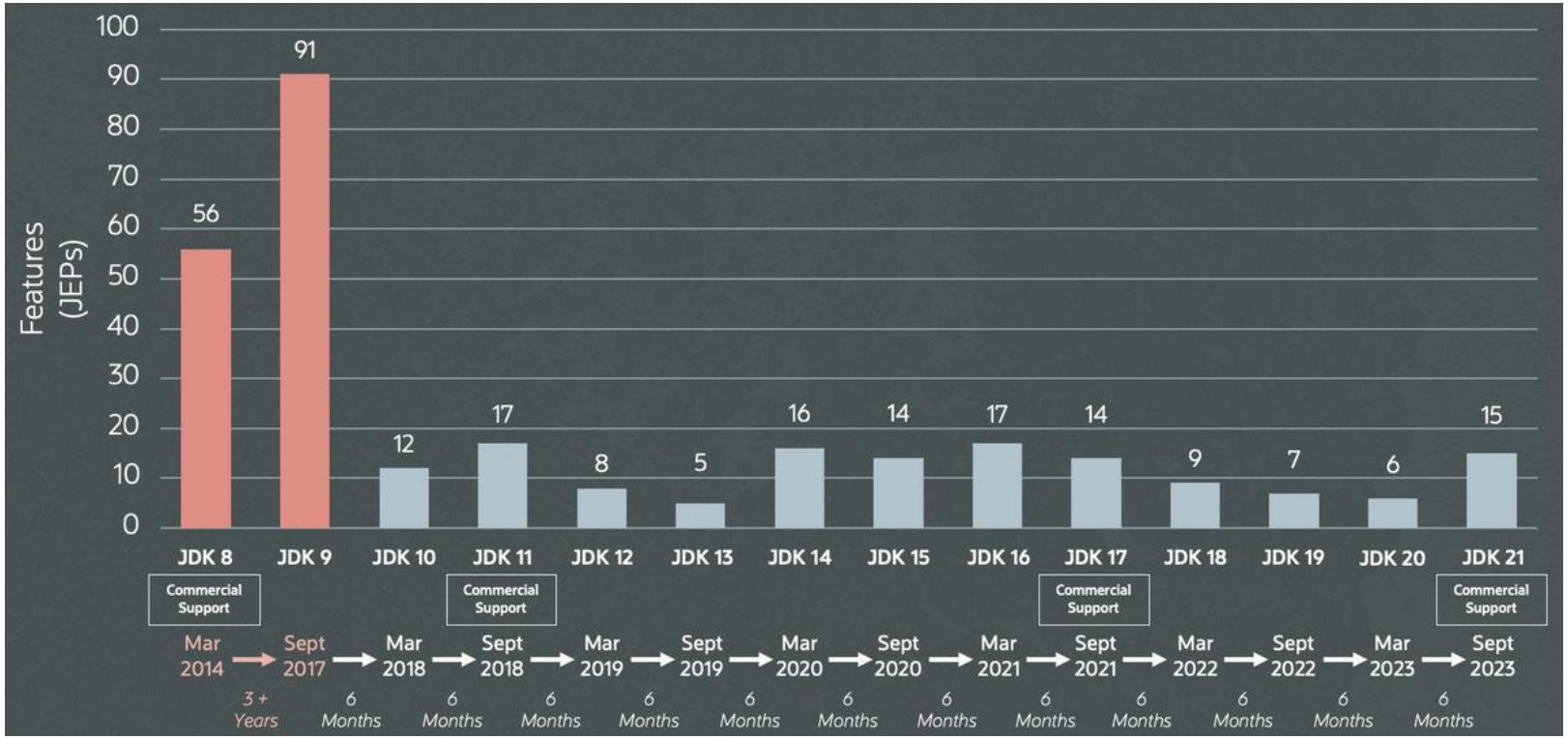
Constant Evolution through JEPs



- The number of JEPs is on the **constant rise**
- Continuous flow of **new features**
- What about Long Term Support (LTS)?

LTS (Long Term Support) Releases

- **Accumulating improvements** over 6-months feature releases
- **New LTS (Long Term Support) release** schedule –> **every 2 years** (instead 3)
- **LTS releases** presented a significant number of JEPs
  - JDK **8** – **56** JEPs
  - JDK 9-**11** – **29** JEPs
  - JDK 12-**17** – **74** JEPs
  - JDK 18-**21** – **38** JEPs



Is Java "Free"?

- Use of **OpenJDK** for **free** with **GPLv2+CE license**
- **Updates** (code patches) – **free of charge**
- **Support** (fixing bugs and answering questions) – it was **never free of charge**

What about **Oracle JDK?**

- **Oracle JDK 11-16** under **Oracle Technology Network (OTN)** license in **production** used with **commercial Java SE subscription**
  - Completely free JDK 11-16 are only OpenJDK binaries
- **Oracle JDK 17+** with **Oracle No-Fee Terms and Conditions (NFTC)** licensing
  - Oracle JDK permits **free use** for all users, even commercial and production use

Current State of Java?

Some questions for all of us:

- Still using **Java 8** (2014)?
- Switched to the "**older**" **Java 11 LTS** (2018)?
- Upgrading to the "**old**" **Java 17 LTS**(2021)?
- Anyone using "**new**" **Java 21 LTS**(2023)?

Some Important Features

OpenJDK

Important features in Java 8-21:

- **Java Platform Module System** and **6-months/2-years OpenJDK** releases
- **Language Features** – Local-Variable Type Inference (vars), Sealed Classes, Hidden Classes, Records, Switch Expressions, Pattern Matching, Text Blocks …
- **Libraries and APIs** – Foreign-Memory Access API, Vector API, Pseudo-Random Generator, Deserealization Filters…
- **Memory Management** – various Garbage Collectors and default G1
- **Easier Debugging and Modernizing Infrastructure**– Flight Recorder, JFT Event Streaming, NullPointerExceptions, Git, ports …
- **Deprecations & Removals** – ~~CMS GC, Nashorn, Biased Locking, RMI Activation, Applet API, Security Manager~~…

Innovation with Incubators and Preview

- **Experimental Features**
  - Test-bed to gather feedback on nontrivial enhancements
- **Incubator Features**
  - New API ideas and tools that, after stabilization, are most likely to be included in JDKs
- **Preview Features**
  - Features believed to be implemented but subject to changes before becoming final
  - **Fully implemented** and **fully specified**, yet **impermanent**, made available in a release to get real world use feedback from developers
  - To try out has to be **enabled** at compile time and at runtime with `--enable-preview`
- **Early Access Releases**
  - Allowing developers to prepare for the next version of JDK in advance

JDK 19

- **JDK 19** in **September 2022**
  - **New features** and **APIs** at openjdk.java.net/projects/jdk/19/
- **JEPs** delivered:
  - JEP 405: **Record Patterns** (Preview)
  - JEP 422: Linux/RISC-V Port
  - JEP 424: **Foreign Function & Memory API** (Preview)
  - JEP 425: **Virtual Threads** (Preview)
  - JEP 426: **Vector API** (Fourth Incubator)
  - JEP 427: **Pattern Matching for switch** (Third Preview)
  - JEP 428: **Structured Concurrency** (Incubator)

JDK 20

- **JDK 20** in **March 2023**
  - **New features** and **APIs** at openjdk.org/projects/jdk/20/
- **JEPs** delivered:
  - JEP 429: **Scoped Values** (Incubator)
  - JEP 432: **Record Patterns** (Second Preview)
  - JEP 433: **Pattern Matching for switch** (Fourth Preview)
  - JEP 434: **Foreign Function & Memory** API (Second Preview)
  - JEP 436: **Virtual Threads** (Second Preview)
  - JEP 437: **Structured Concurrency** (Second Incubator)
  - JEP 438: **Vector API** (Fifth Incubator)

JDK 21

- **JDK 21** released on **September 19, 2023**
  - **New features** and **APIs** at openjdk.org/projects/jdk/21/
- **JEPs** delivered:
  - JEP 430: **String Templates** (Preview)
  - JEP 431: **Sequenced Collections**
  - JEP 439: **Generational ZGC**
  - JEP 440: **Record Patterns**
  - JEP 441: **Pattern Matching for switch**
  - JEP 442: **Foreign Function & Memory API** (Third Preview)
  - JEP 443: **Unnamed Patterns and Variables** (Preview)
  - JEP 444: **Virtual Threads**
  - JEP 445: **Unnamed Classes and Instance Main Methods** (Preview)
  - JEP 446: **Scoped Values** (Preview)
  - JEP 448: **Vector API** (Sixth Incubator)
  - JEP 449: Deprecate the Windows 32-bit x86 Port for Removal
  - JEP 451: Prepare to Disallow the Dynamic Loading of Agents
  - JEP 452: Key Encapsulation Mechanism API
  - JEP 453: **Structured Concurrency** (Preview)

JDK 22 – Foreseeable Future

- **JDK 22** to be released in **March 2024**
  - **New features** and **APIs** at openjdk.org/projects/jdk/22/
- **JEPs** targeted (so far):
  - JEP 454: **Foreign Function & Memory API**
- **JEPs** not targeted (yet):
  - JEP 455: **Primitive types in Patterns, instanceof, and switch** (Preview)
  - JEP 456: **Unnamed Variables and Patterns**
  - JEP 457: **Class-File API** (Preview)
  - JEP 458: **Launch Multi-File Source-Code Programs**
  - JEP 459: **String Templates** (Second Preview)
  - JEP 460: **Vector API** (Seventh Incubator)
  - JEP draft: **Null-Restricted Value Class Types** (Preview)

Virtual Threads

- JEP 444: **Virtual Threads** https://openjdk.org/jeps/444
  - Previously previewed in JDK 20 and 19, finalized in JDK 21
- Virtual threads are **lightweight JVM threads** that dramatically reduce the effort of writing and maintaining high-throughput concurrent applications
- Server applications written in the **simple thread-per-request** style to scale with near-optimal hardware utilization
- Existing code using **java.lang.Thread API** could **adopt** with minimal changes
- Easy **debugging** and **profiling** with current JDK tools
- Now with **guaranteed support** for **thread-local variables**
  - Ensures that more existing libraries can be used unchanged with virtual threads and assists with migrating task-oriented code to use virtual threads
- Developers can choose whether to use **virtual threads** or **platform threads**

Virtual Threads

- Updated and new java.lang.Thread API:
  - Thread.**Builder**, methods **ofVirtual**(), **ofPlatform**(), **isVirtual**(), **startVirtualThread**(Runnable), **getAllStackTraces**()
- *Example*: Creates a new unstarted virtual thread named "duke"

```
Thread thread =
Thread.ofVirtual().name("duke").unstarted(runnable);
```

- *Example*: Starts a virtual thread

```
public class Main {
    public static void main(String[] args)
      throws InterruptedException {
        var vThread = Thread.startVirtualThread(() -> {
            System.out.println("Hello from virtual thread");
        });
        vThread.join();
    }
}
```

Virtual Threads – Example

- *Example*: Obtains an ExecutorService that will create a new virtual thread for each submitted task, and then it submits 10000 tasks and waits for all of them to complete:

```
try (var executor =
Executors.newVirtualThreadPerTaskExecutor()) {
    IntStream.range(0, 10000).forEach(i -> {
        executor.submit(() -> {
            Thread.sleep(Duration.ofSeconds(1));
            return i;
        });
    });
} // executor.close() called implicitly, and waits
```

- JDK runs it on a small number of OS threads, perhaps as few as one

Virtual Threads – More …

- *More examples*:
  - Nicolai Parlog github.com/nipafx/loom-lab
  - Bazlur Rahman github.com/rokon12/project-loom-slides-and-demo-code
- Recommended to watch:
  - **Java 21 new feature: Virtual Threads** - youtu.be/5E0LU85EnTI
  - **Virtual Threads and Structured Concurrency in Java 21 With Loom** - www.youtube.com/live/QxxG66eQoTc
  - **Java 21: Focus on Virtual Threads and Pattern Matching** - www.youtube.com/watch?v=d_XmNicqC2I
  - **Virtual Thread Deep Dive** - Inside Java Newscast #23 https://youtu.be/6dpHdo-UnCg
  - **Launching 10 millions virtual threads with Loom** - JEP Café #12 youtu.be/UVoGE0GZZPI
  - **Java Asynchronous Programming Full Tutorial with Loom and Structured Concurrency** - JEP Café #13 youtu.be/2nOj8MKHvmw

Structured Concurrency

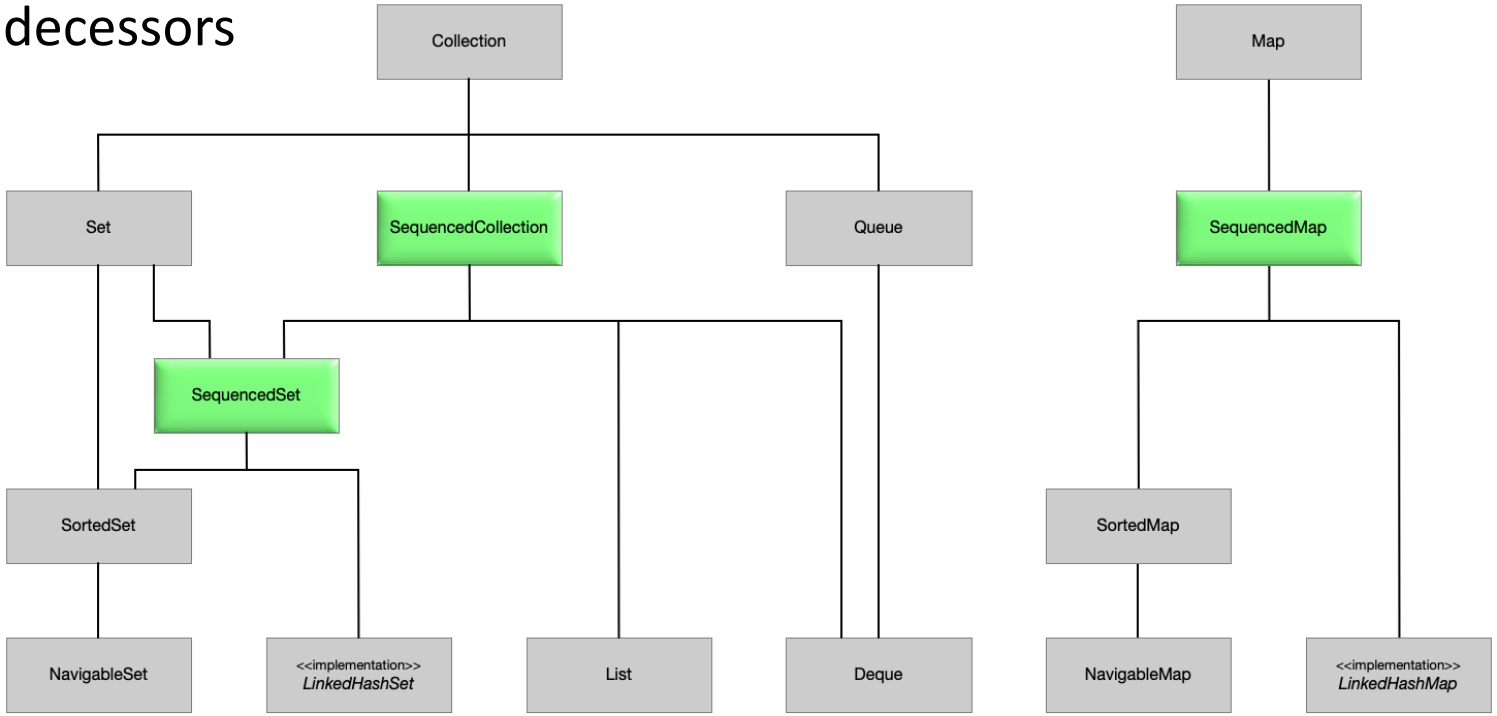
- JEP 453: **Structured Concurrency** (Preview) openjdk.org/jeps/453
  - The term was coined by Martin Sústrik and popularized by Nathaniel J. Smith
- Simplify multithreaded programming by **structured concurrency API** – treat multiple tasks running in different threads as a single unit of work
  - Streamlining error handling and cancellation, improving reliability, and enhancing observability in server applications
- In structured concurrency, **subtasks work on behalf of a task** – the task awaits the subtasks' results and monitors them for failures
- The power of structured concurrency for multiple threads comes from:
  a) Well-defined **entry and exit points** for the flow of execution
  b) A strict **nesting of the lifetimes of operations**
- Structured concurrency is a **great match for virtual threads**

Sequenced Collections

- JEP 432: **Sequenced Collections** https://openjdk.org/jeps/431
- Collections framework lacks a collection type to represent a sequence of elements with a defined **order** and using a uniform set of **operations**
- List and Deque both define an encounter order but their common supertype Collection does not
- SortedSet and LinkedHashSet both define an encounter order but Set and HashSet do not
- Neither Collection (too general) nor List (too specific) can describe a parameter or return value that has an encounter order
- New interfaces developed to represent **collections** with a defined encounter **order** (well-defined from first up to the last element)

Sequenced Collections

- Retrofitting these new interfaces into the existing collections type hierarchy
  - Uniform **APIs** for accessing **first** and **last** elements, and processing elements in **reverse order**
- **Sequenced collection's** elements have a defined encounter order
  - All elements have successors/predecessors
  - Supports operations at either end, processing forward and reverse
- **Sequenced set** is a sequenced Set with no duplicate elements
- **Sequenced map** is a Map whose entries have a defined encounter order



Sequenced Collections – Example

- *Example*: SequencedCollection interface methods

```
interface SequencedCollection<E> extends Collection<E> {
    SequencedCollection<E> reversed();        // new method
    void addFirst(E);                         // methods promoted from
Deque
    void addLast(E);
    E getFirst();
    E getLast();
    E removeFirst();
    E removeLast();
}
```

- *Example*: SequencedSet methods

```
interface SequencedSet<E> extends Set<E>, SequencedCollection<E> {
    SequencedSet<E> reversed();          // covariant override
}
```

Sequenced Collections – Example

- *Example*: SequencedMap methods

```
interface SequencedMap<K,V> extends Map<K,V> {
    SequencedMap<K,V> reversed();                    // new methods
    SequencedSet<K> sequencedKeySet();
    SequencedCollection<V> sequencedValues();
    SequencedSet<Entry<K,V>> sequencedEntrySet();
    V putFirst(K, V);
    V putLast(K, V);
    Entry<K, V> firstEntry();    // methods from NavigableMap
    Entry<K, V> lastEntry();
    Entry<K, V> pollFirstEntry();
    Entry<K, V> pollLastEntry();
}
```

String Templates

- JEP 430**: String Templates** (Preview) openjdk.org/jeps/430
  - JEP 459: String Templates (Second Preview) planned openjdk.org/jeps/459
- Coupling literal text with **embedded expressions** and **template processors**
  - **Interpreted at run time**, possibly after further validation and transformation
  - Making it easy to express **strings that include values computed at run time**
  - Complement Java's existing string literals and text blocks with enhanced readability
- **Simplify** expressing strings that include **values computed at runtime**
  - Retain **flexibility** – Java libraries to define the formatting syntax
  - Improve the **security** – supports validation and transformation
- Simplify the use of APIs with strings in other languages (e.g., SQL, XML, JSON)
  - Develop non-string expressions combining literal text and embedded expressions
  - Does not change string concatenation (+), StringBuilder, and StringBuffer

String Templates

- **STR** is a template processor performing **string interpolation** by replacing embedded expression with the (stringified) value of that expression
- *Example*: String template

```
String name = "Branko";
String info = STR."My name is \{name}";
assert info.equals("My name is Branko");   // true
```

- String template expression **`STR."My name is \{name}"`** consists of:
  - Template processor (**`STR`**) and dot character (`.`)
  - Template (**`"My name is \{name}"`**) contain embedded expression (**`\{name}`**)
- When a template expression is evaluated at run time, literal text is combined with the values of the embedded expressions in order to produce a result

String Templates – Examples

- *Example*: Embedded expressions can be strings

```
String firstName = "John";
String lastName  = "Smith";
String sortName  = STR."\{lastName}, \{firstName}";
```

- *Example*: Embedded expressions can perform arithmetic

```
int x = 10, y = 20;
String s = STR."\{x} + \{y} = \{x + y}";
```

- *Example*: Embedded expressions are evaluated (left to right)

```
int index = 0; // embedded expressions can be postfix increment
String data = STR."\{index++}, \{index++}, \{index++} ";
```

- *Example*: Embedded expressions can invoke methods and access fields

```
String s = STR."You have a \{getOfferType()} waiting for you!";
```

Record Patterns

- JEP 440: **Record Patterns** openjdk.org/jeps/440 – deconstruct record values
  - Since JDK 16 we have **record classes** as transparent carriers for data
  - Based on **Pattern Matching for switch** and **type patterns**
- **Record patterns** and **type patterns** can be nested to enable a powerful, declarative, and composable form of data navigation and processing
- **Declaration of local variables** for extracted components in **patterns**
- Initializes those variables by invoking the accessor methods when a value is **matched** against the pattern
- In effect, a record pattern disaggregates an instance of a record into its components

Record Patterns

- *Example*: The record pattern tests whether a value is an instance of Point, and also **extracts** components (x and y) from the value directly

```
record Point(int x, int y) {}
void printSum(Object o) {
    if (o instanceof Point(int x, int y)) {
        System.out.println(x+y);
    }
}
```

- *Example*: Inference of type arguments in **instanceof** expressions and **switch** statements and expressions works with **nested** record patterns

```
record Box<T>(T t) {}
static void test1(Box<Box<String>> bbs) {
    if (bbs instanceof Box<Box<String>>( Box( var s))) {
        System.out.println("String " + s);
    }
}
```

Foreign Function & Memory API

- JEP 454: **Foreign Function & Memory API** https://openjdk.org/jeps/454
  - Currently in 3rd Preview, evolved from JDK 18, 19, and 20 , coming in JDK 22
- API for statically-typed, pure-Java access to **native code**
- Java call to **native libraries** and **process native data** without the brittleness of JNI (Java Native Interface)
- Efficiently invoking foreign functions (outside of JVM) and by safely accessing foreign memory (not managed by JVM)
  - **Foreign Linker** API supports foreign function support
  - **Foreign Memory Access** API allows access to memory outside of heap
- *Examples*: Carl Dea (Azul) github.com/carldea/panama4newbies

Unnamed Classes and Instance Main Methods

- JEP 445: **Unnamed Classes and Instance Main Methods** (Preview) openjdk.org/jeps/445
- Write first programs without needing to understand complex language features
- *Example*: Can we make this simpler?

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

- YES! Optional **public, static, args**, and **unnamed classes** ("main class") with **automatic import** of static methods

```
void main() {
    println("Hello, World!");
}
```

Scoped Values

- JEP 446: **Scoped Values (Preview)** openjdk.org/jeps/446
- Scoped values may be **safely and efficiently shared to methods** without using method parameters
- A common pattern to **transfer control** from one component (e.g., "framework") to another (e.g., "application code") and then back
- Preferred to thread-local variables, especially when using large numbers of virtual threads
- In effect, a scoped value is an **implicit method parameter** - as if every method in a sequence of calls has an additional **invisible** parameter
  - Only the methods that have access to the scoped value object can access its value (the data)
- Possible to **pass data securely** from a caller to a faraway callee through intermediate methods that **do not declare a parameter** and have no access to the data

Vector API

- JEP 460: **Vector API** (7th Incubator) openjdk.org/jeps/460
  - Proposed in JDK 16, upgraded in JDK 17-21
- API for **vector computations** that reliably compile at runtime to optimal vector instructions on supported CPU architectures
  - Achieving performance superior to equivalent scalar computations
  - The latest incarnation includes performance enhancements and bug fixes
- Allows developers to **write complex vector algorithms** directly in Java

Primitive types in Patterns, instanceof, and switch

- JEP 455: **Primitive types in Patterns, instanceof, and switch (Preview)** openjdk.org/jeps/455
  - Promoted to Candidate status
- **Enhances pattern matching** by allowing primitive type patterns used in **all** pattern contexts
- Aligns the semantics of primitive type patterns with **instanceof**
- *Example*:

```
record Customer(String name, int age) { }
if (json instanceof JsonObject(var map)
    && map.get("name") instanceof JsonString(String name)
    && map.get("age") instanceof JsonNumber(int age))
{
    return new Customer(name, age);     // no cast of age
}
```

- Extends also **switch** to allow primitive constants as case labels

Unnamed Variables and Patterns

- JEP 456: **Unnamed Variables and Patterns** openjdk.org/jeps/456
  - Promoted to Target status to finalize it from JEP 443
- **Unnamed variables** can be initialized but not used
- **Unnamed patterns** match a record component without stating the component's name or type
- Both of these are denoted by the **underscore (_)** character
- The following declarations can introduce an **unnamed variable**:
  - A local variable declaration statement in a block
  - The resource specification of a try-with-resources statement
  - The header of a basic for loop
  - The header of an enhanced for loop
  - An exception parameter of a catch block
  - A formal parameter of a lambda expression

Unnamed Variables – Examples

- *Example*: An enhanced for loop with side effects

```
static int count(Iterable<Order> orders) {
    int total = 0;
    for (Order _ : orders)        // unnamed variable
        total++;
    return total;
}
```

- *Example*: Initialization of a for loop can declare unnamed local variables

```
for (int i = 0, _ = sideEffect(); i < 10; i++) { ... i ... }
```

- *Example*: In lambda whose parameter is irrelevant

```
stream.collect(Collectors.toMap(String::toUpperCase, _ -> "NODATA"))
```

- *Example*: In try-with-resources:

```
try (var _ = ScopedContext.acquire()) {
    ... no use of acquired resource ...
}
```

Unnamed Variables – Examples

- *Example*: An assignment where the result of the expression is not needed

```
Queue<Integer> q = ... // x1, y1, z1, x2, y2, z2, ...
while (q.size() >= 3) {
    var x = q.remove();
    var y = q.remove();
    var _ = q.remove();      // Unnamed variable
    ... new Point(x, y) ...
}
```

- *Example*: A catch block

```
String s = ...
try {
    int i = Integer.parseInt(s);
    ... i ...
} catch (NumberFormatException _) {
    System.out.println("Bad number: " + s);
}
```

Class-File API

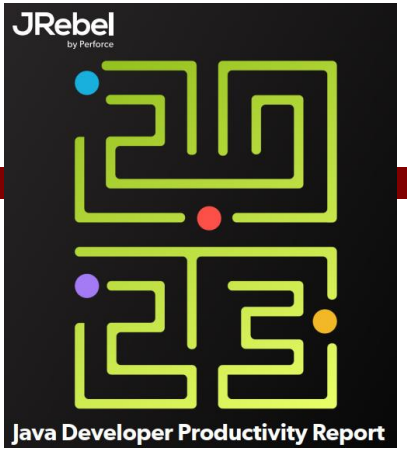
- JEP 457: **Class-File API (Preview)** openjdk.org/jeps/457
  - Promoted to Candidate status
- Provide an **API** for **reading, writing**, and **transforming** Java **class files**
- Initially as an internal **replacement for ASM** (Java bytecode manipulation and analysis framework) in the JDK with plans to open as a public API
- Brian Goetz (Java language architect at Oracle) characterized ASM as "*an old codebase with plenty of legacy baggage*" and provided background information on how this draft will evolve and ultimately replace it

Projects – Longer-term Java Future

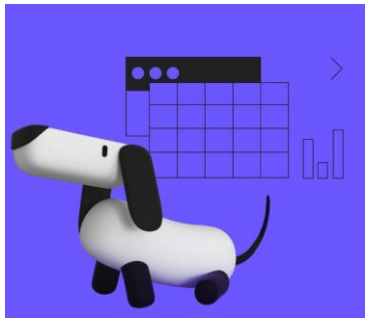
OpenDJK Projects – openjdk.java.net/projects/

- Project **Amber** – incubator for smaller, productivity-oriented **language features** and **simplifying syntax**
- Project **Valhalla** – incubator for **advanced JVM and language feature** candidates
- Project **Loom** – to **increase performance** and **reduce complexity** in concurrent applications
- Project **Panama** – to interconnect JVM and **native** code
- Project **Metropolis** – JVM re-written in Java, i.e. "***Java on Java***"
- Project **Leyden** – improve **start-up time** to achieve peak performance

Some surveys

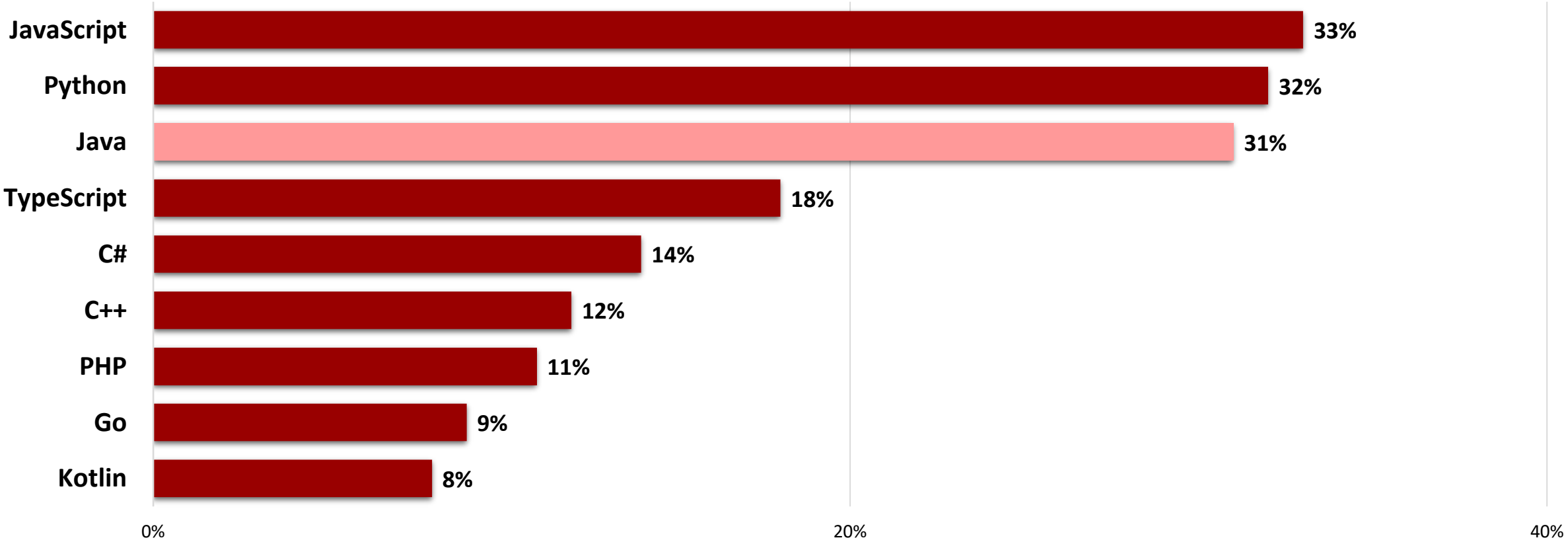


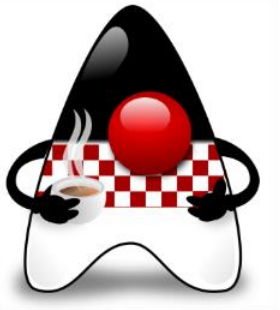
- **Developer Skills Report 2023** by **HackerRank**
  - HackerRank platform data
- **Java Developer Productivity Report 2023** by **JRebel**
  - 411 developers, Nov 2022 – Jan 2023
- **Developer Survey Report 2023** by **Jakarta EE**
  - 2204 developers
- **The State of Developer Ecosystem 2022** by **JetBrains**
  - 29269 developers from 187 countries in 2022
- **2023 Developer Survey** by **Stack Overflow**
  - 67237 developers in May 2023
- **2023 State of the Java Ecosystem** by **New Relic**



Top Languages

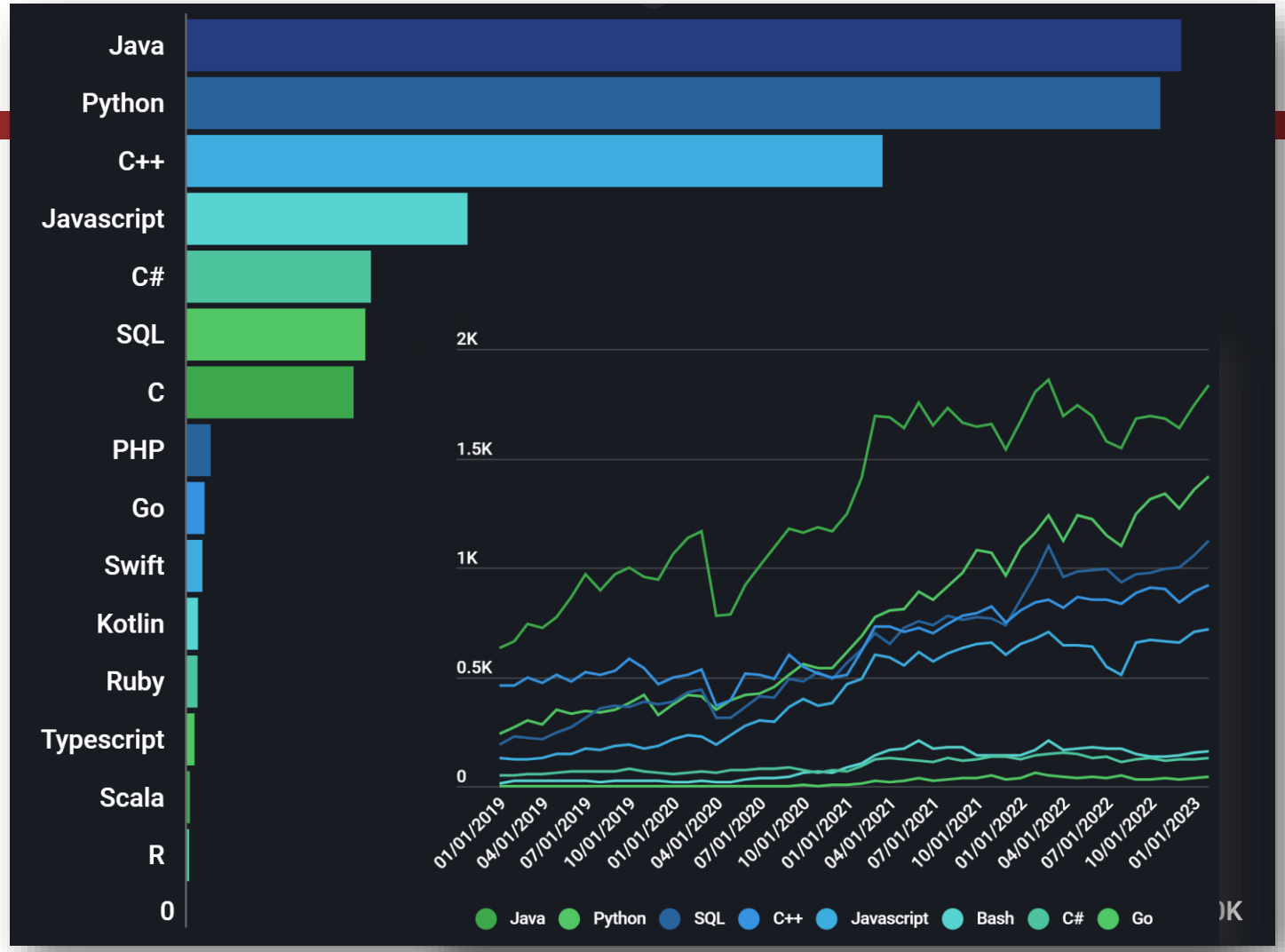
- What is your **primary programming language** (JetBrains)?





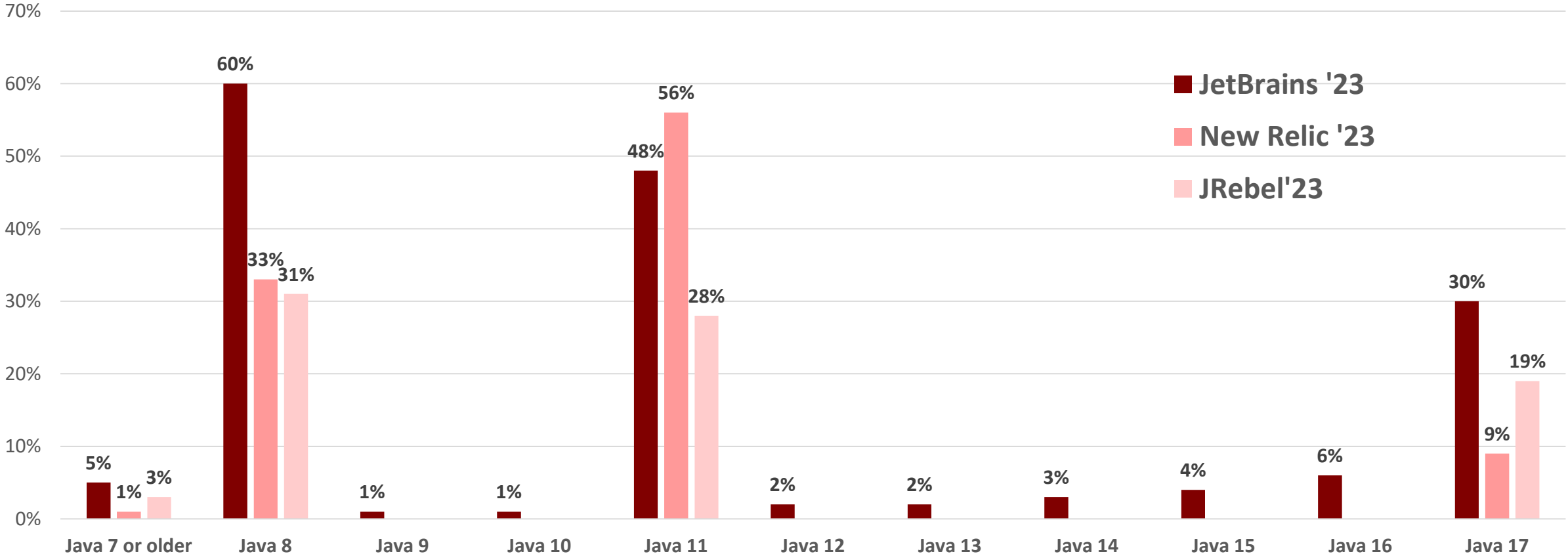
Top Languages

- **Languages preferred** when having multiple options, by number of developers? (HackerRank)
  - **Java** **558 832**
  - Python 547 018
  - C++ 390 921
  - JS 157 660
  - C# 103 397
  - SQL 100 232
  - C 93 636
  - PHP 13 374
  - Go 9 899
- ***"Java, Python, and SQL likely to extend their lead in 2023"***



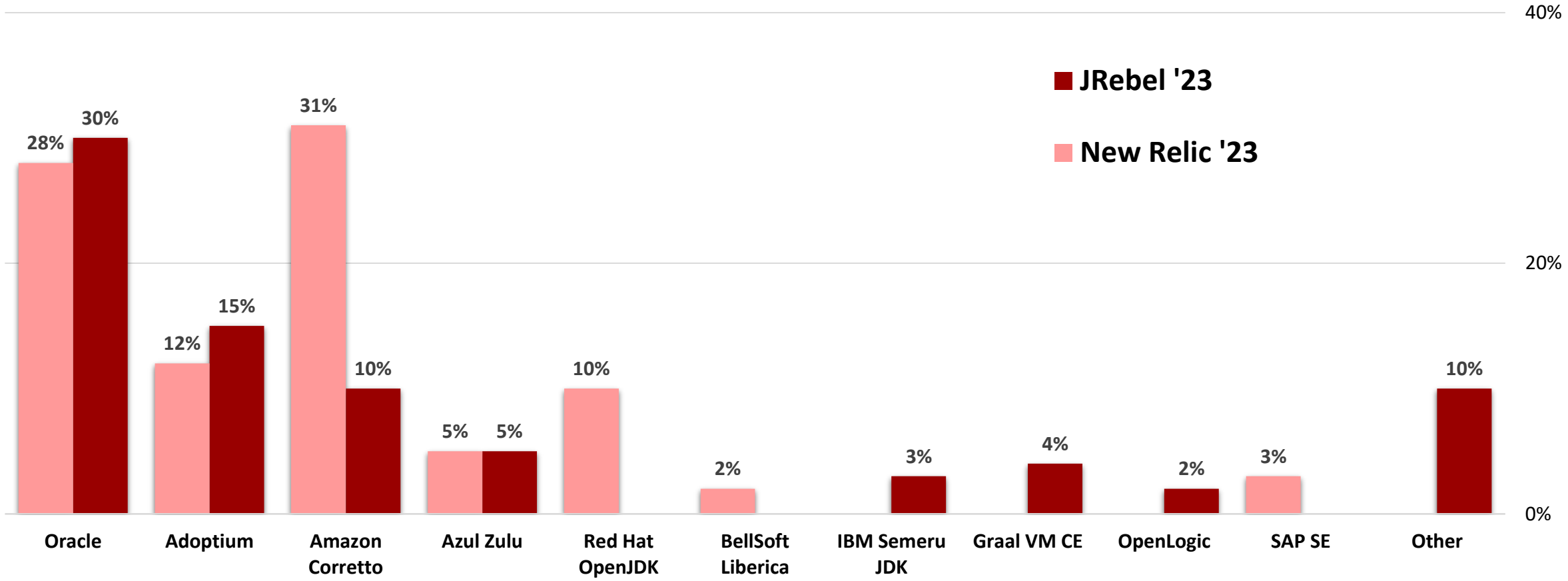
Versions of Java

- **Java platform versions** used in projects? (JetBrains, New Relic, JRebel)



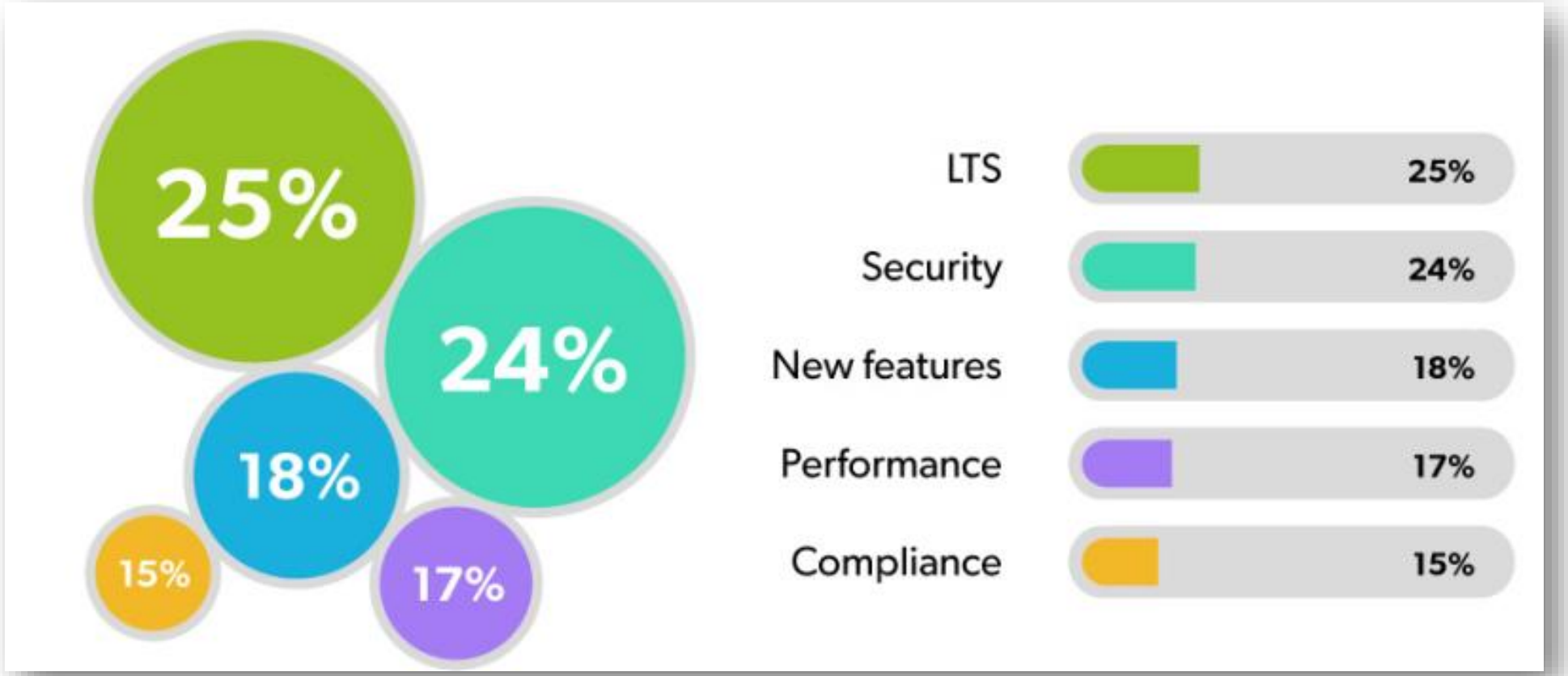
JDK Distributions

- Which **JRE/JDK distribution** do you use? (JRebel, New Relic)



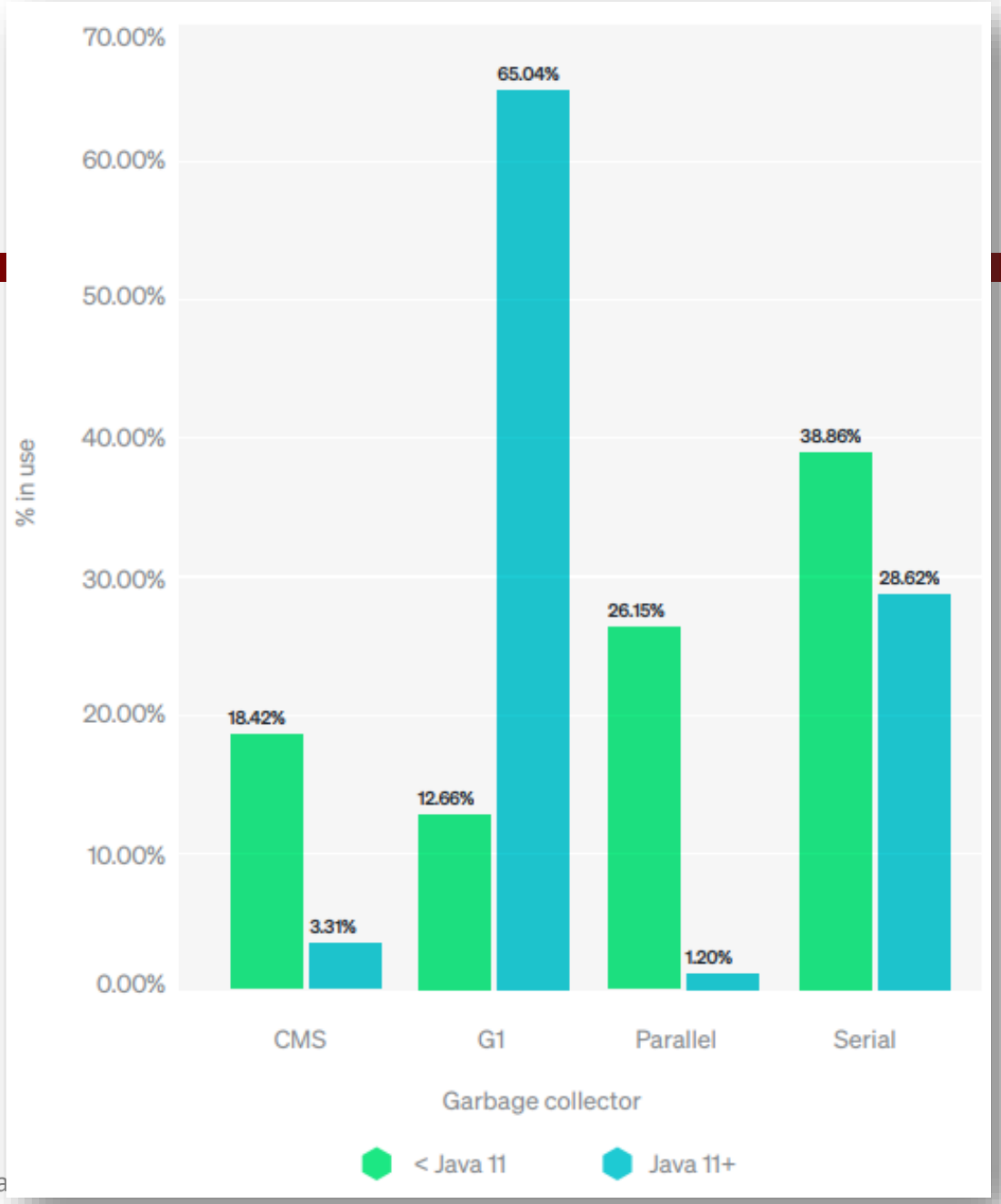
Upgrade Factors

- Which **factors** influence your decision to **upgrade** JDK versions? (JRebel)



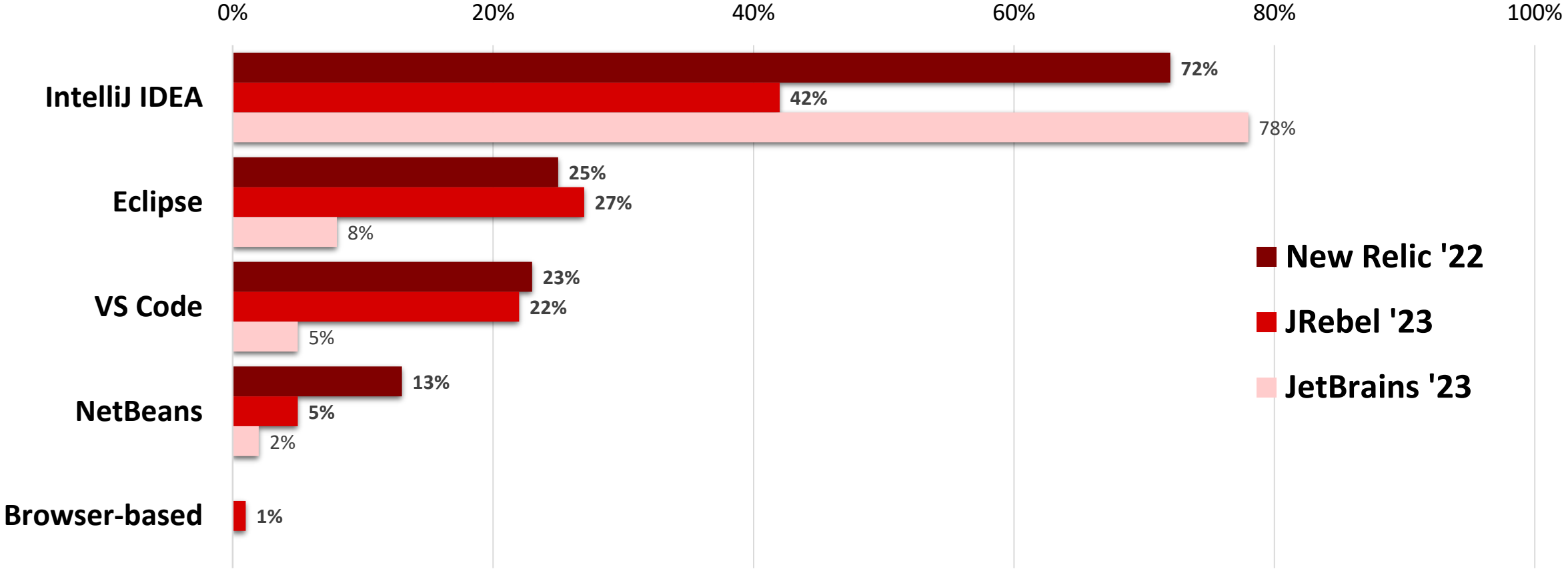
Garbage Collectors

- **Garbage collectors** used? (New Relic)
- *Data shows that the **Garbage-First (G1)** garbage collector continues to be a clear favorite for those using Java 11 or later versions, with 65% of customers using it.*
- *Other experimental garbage collectors that have appeared post-Java 8 (**ZGC** and **Shenandoah**) still show small usage in production systems.*



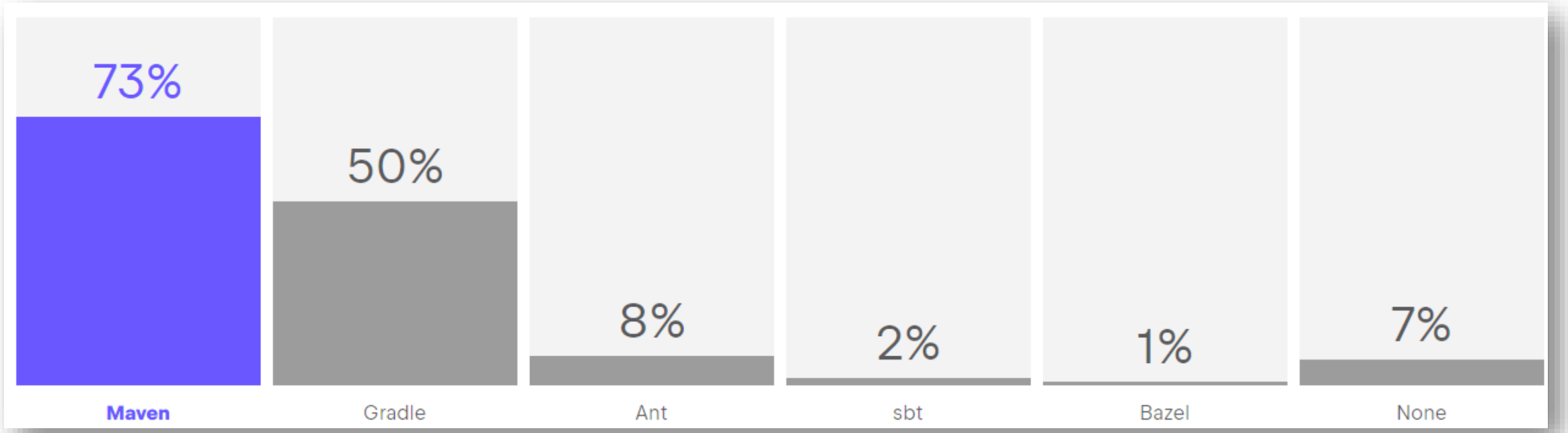
Most popular IDEs

- The most **popular IDEs** used? (New Relic, JRebel, JetBrains)



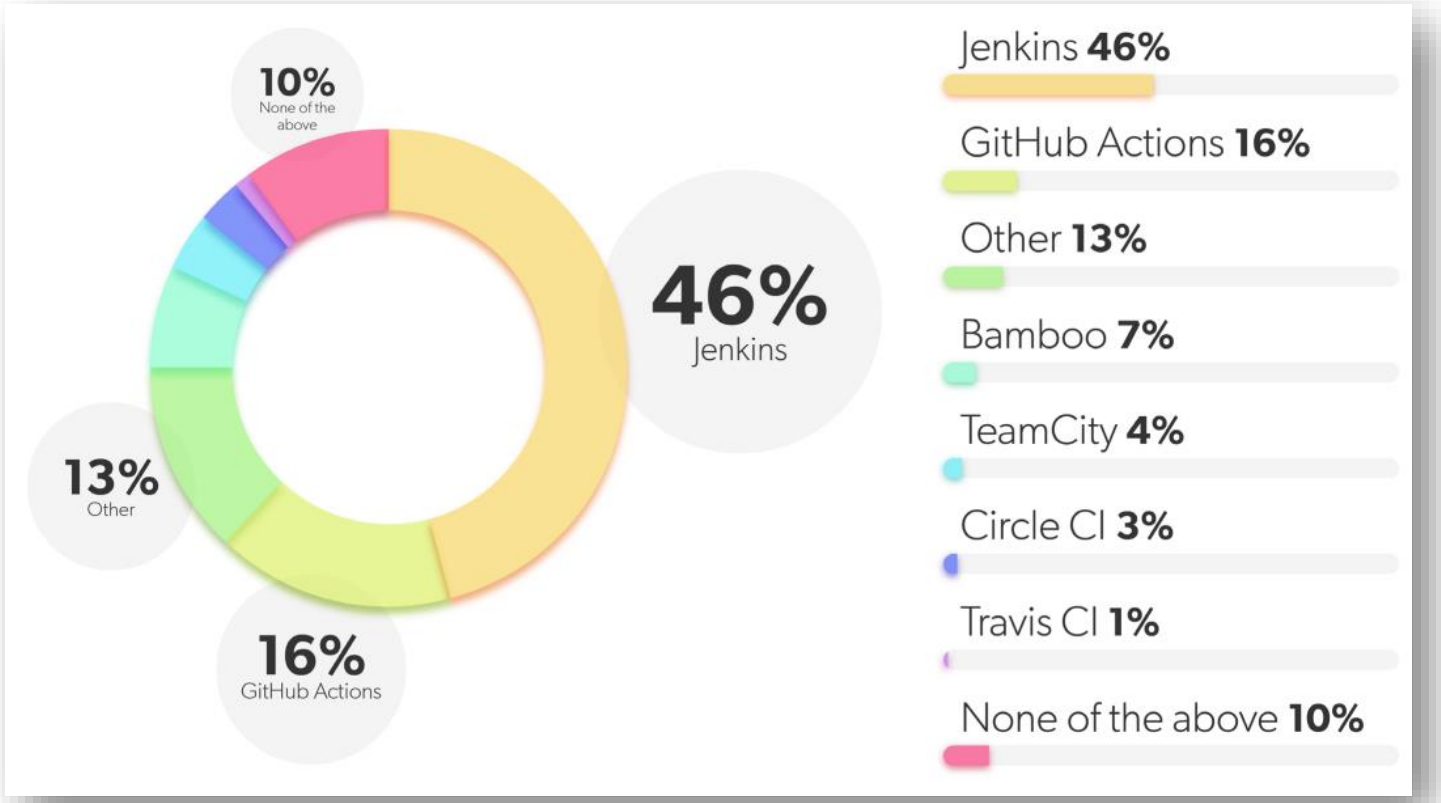
Build Tools

- Which **build systems** do you regularly use? (JetBrains)

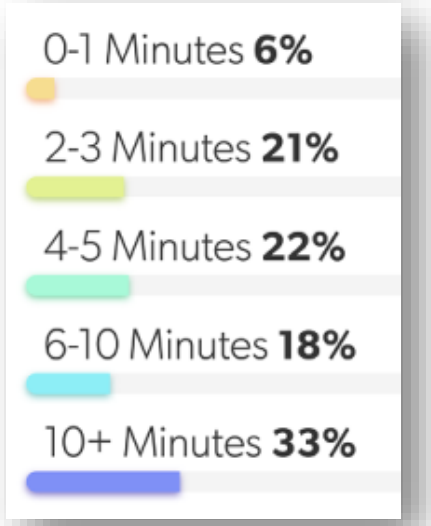


CI/CD

- Which **CI/CD technologies** are you using? (JRebel)

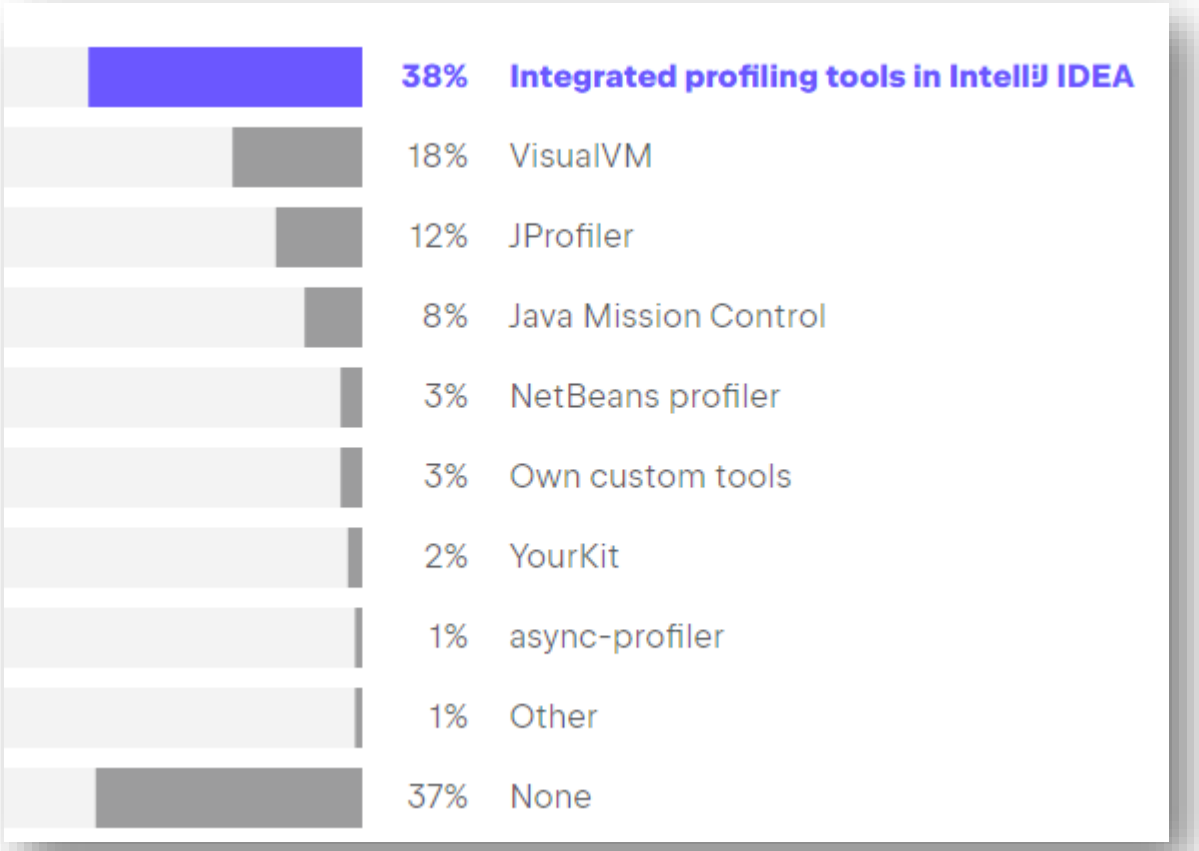


- **How long** does it take to complete CI/CD build?



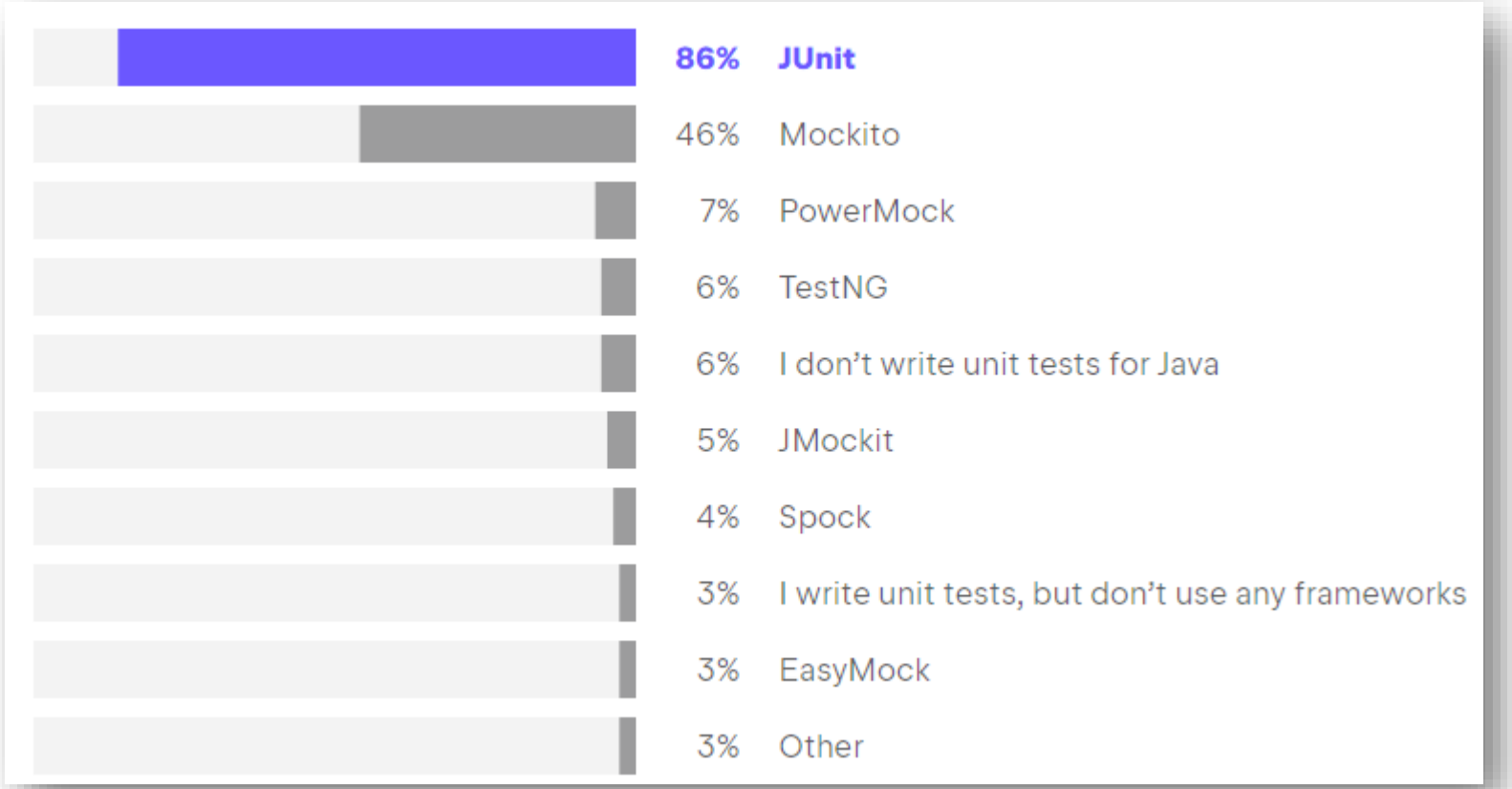
JVM Profilers

- Which **JVM profilers** do you regularly use? (JetBrains)



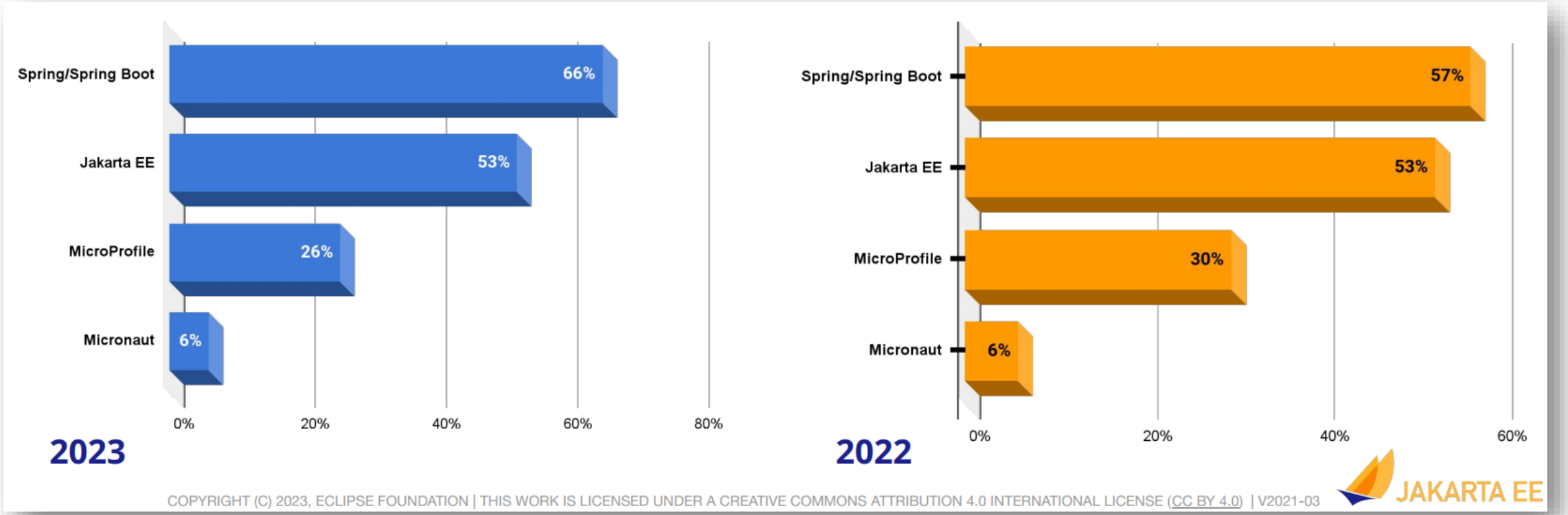
Unit Testing

- Which **unit-testing frameworks** do you use? (JetBrains)



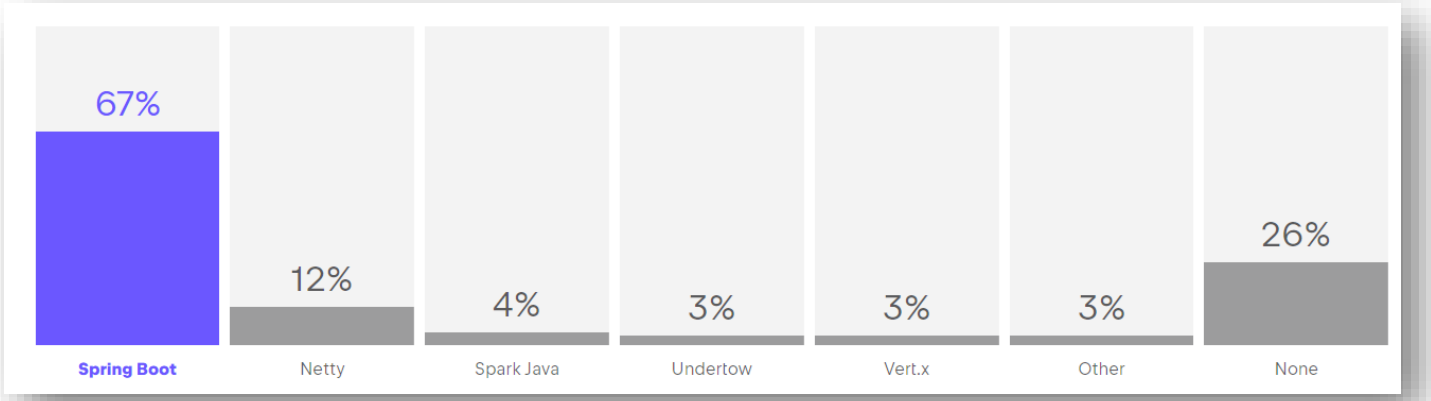
Frameworks

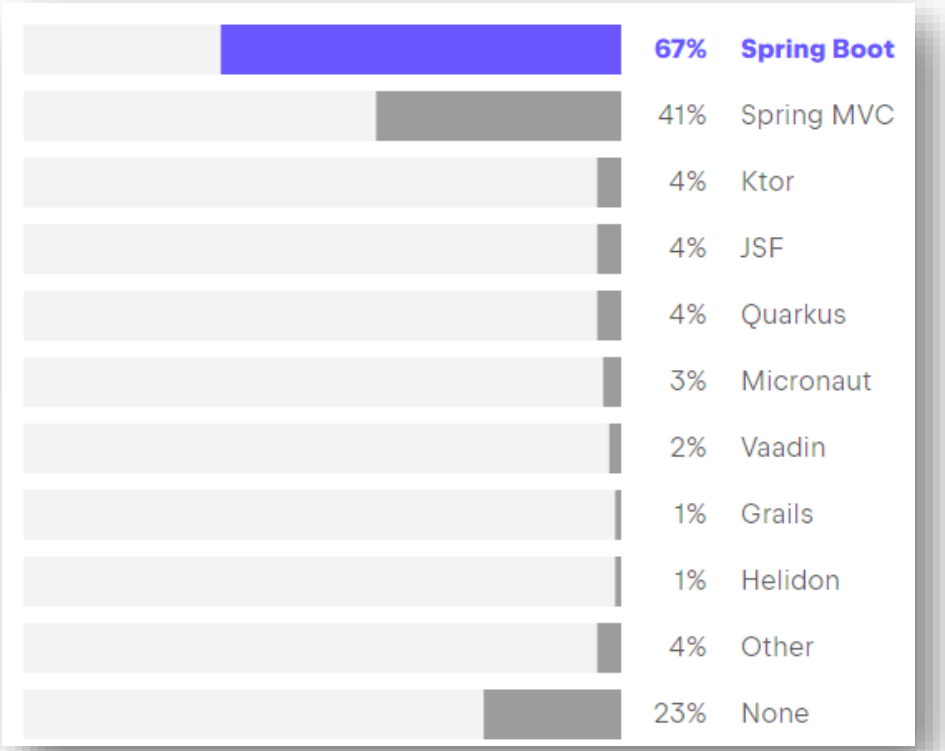
- Top app **frameworks** for building cloud native application? (JakartaEE)



Frameworks

- Which **frameworks** do you use (as alternative to an app server)? (JetBrains)

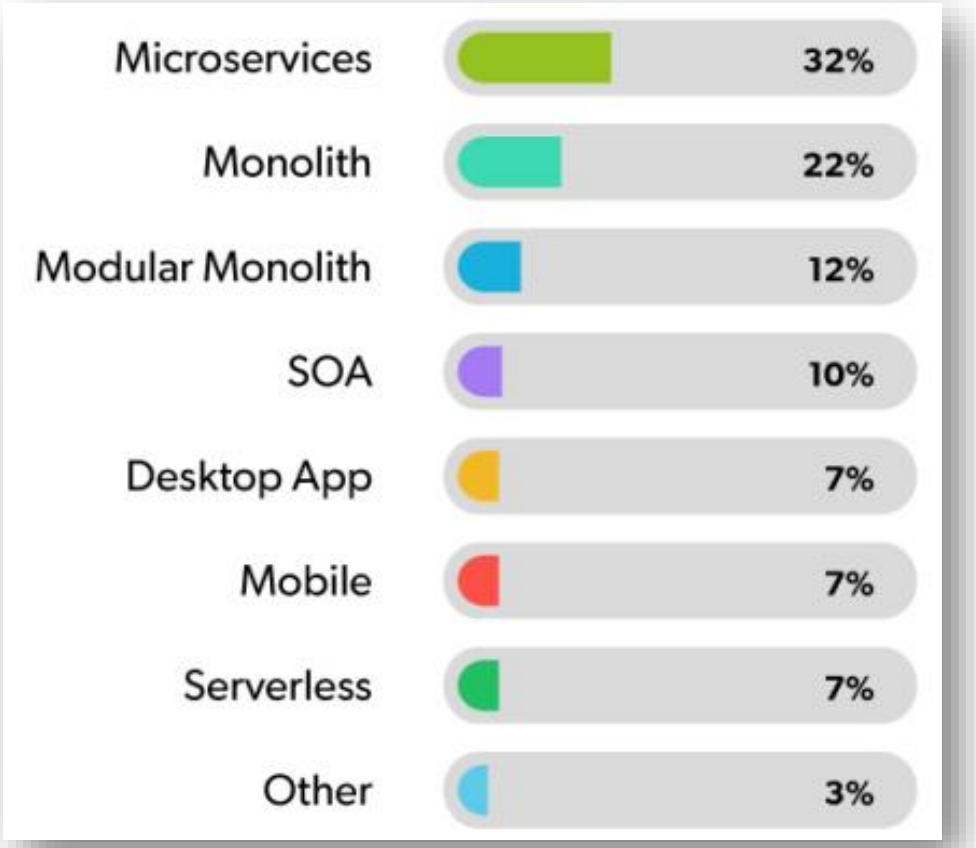




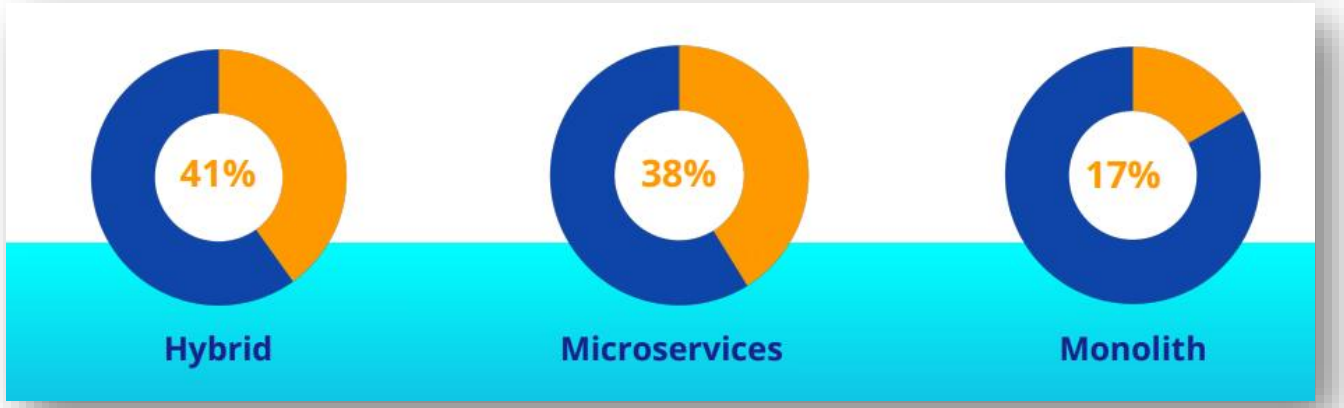
- What **web frameworks** do you use? (JetBrains)

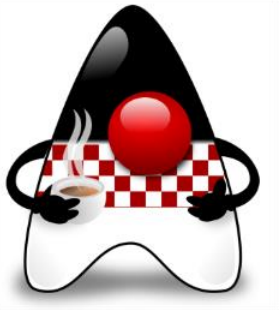
Architecture Trends

- What is the **architecture** of the main application you develop? (JRebel)



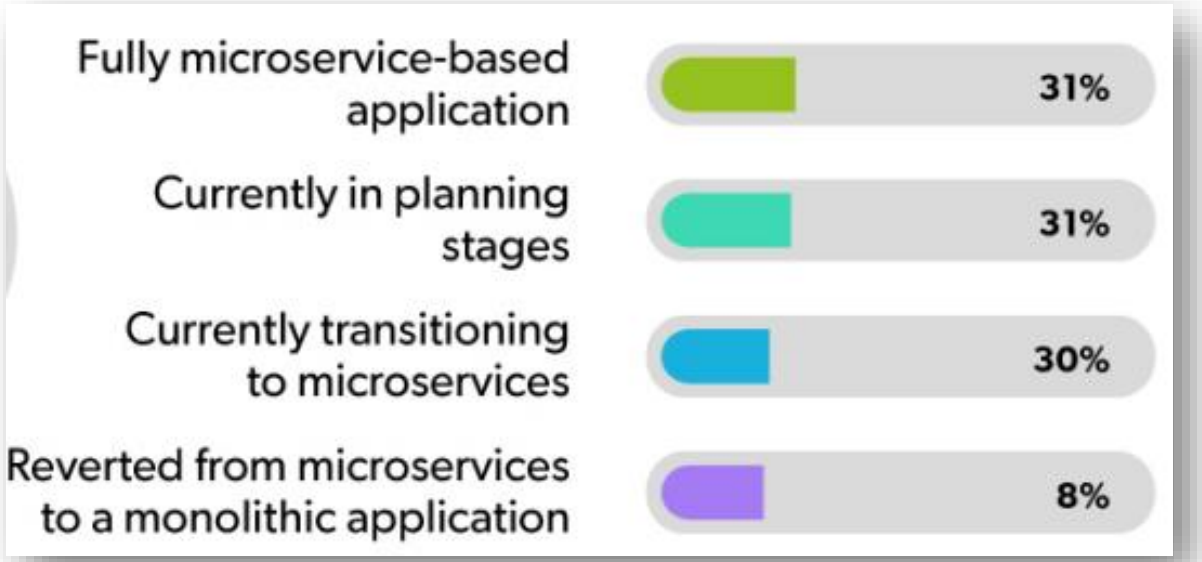
- ¸Top **architectural approaches** for implementing Java systems in the cloud? (Jakarta EE)

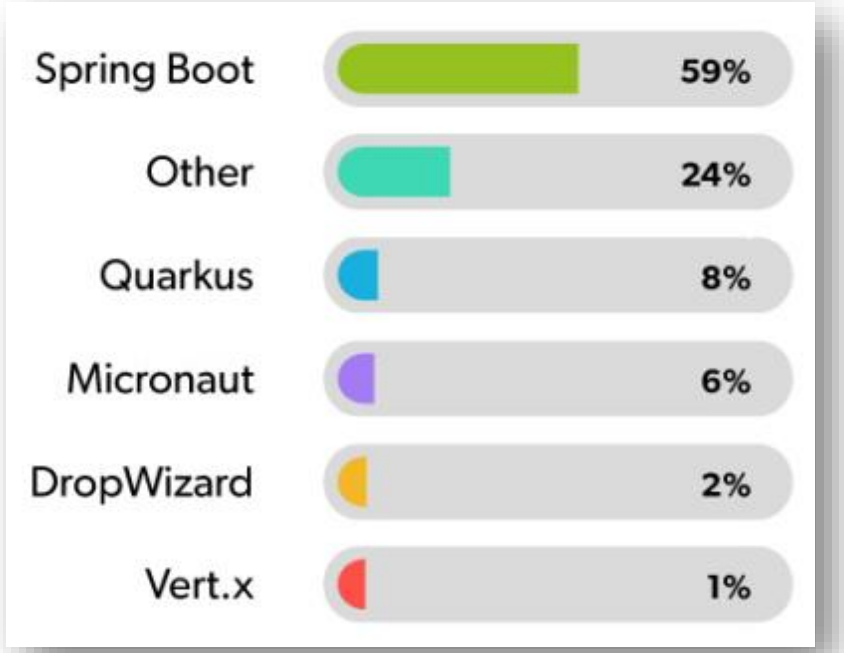




Microservices

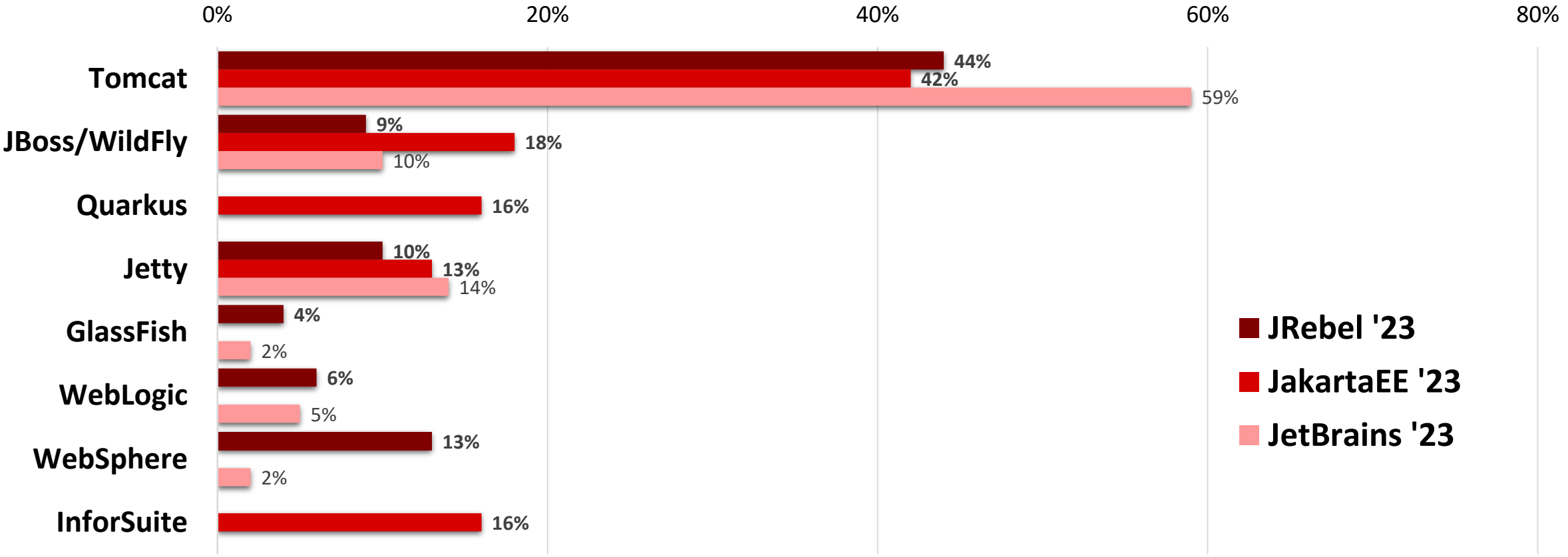
- What is your status for **Microservice** adoption? (JRebel)
- What **Microservice Application Framework** on your main project? (JRebel)





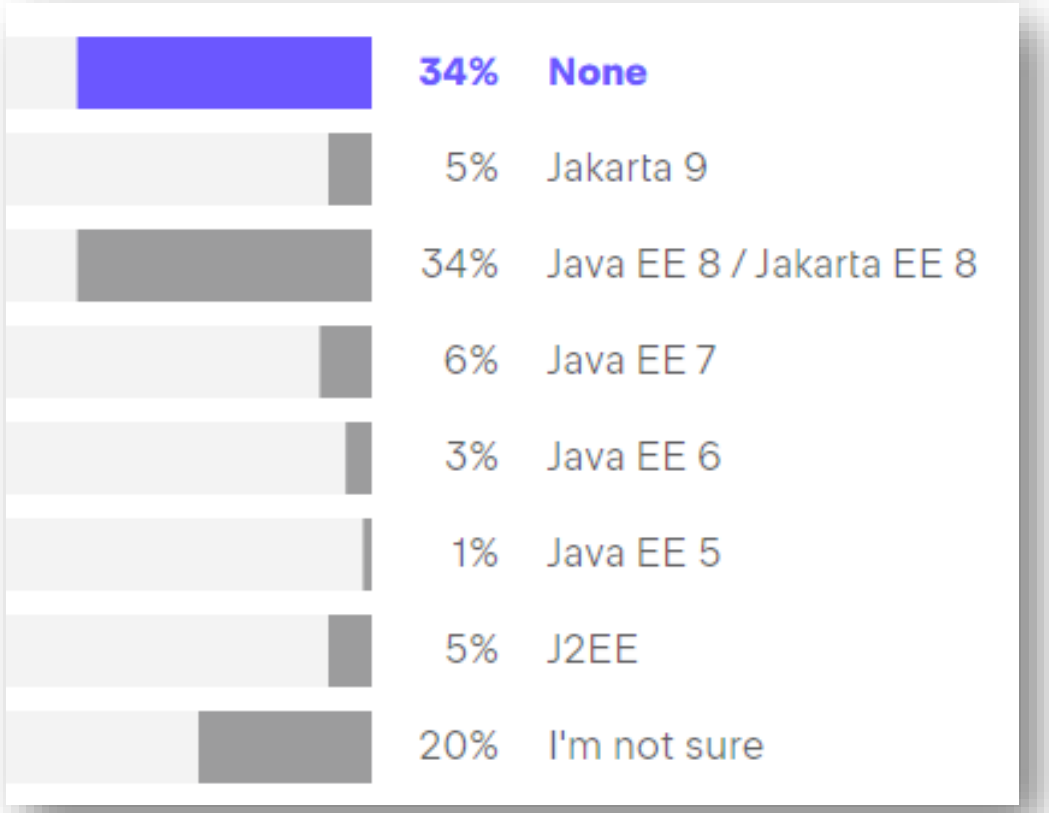
App Servers

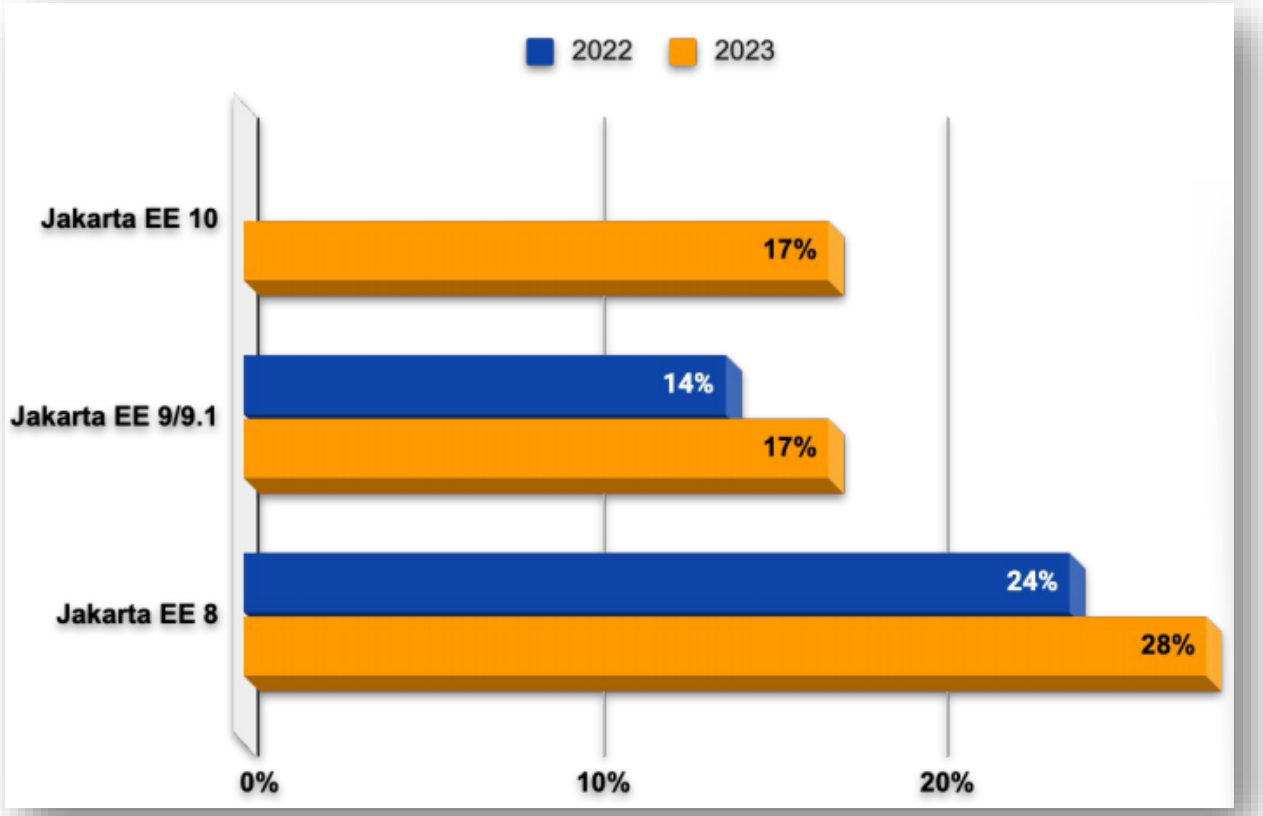
- What **Application Server** do you use on your main application? (JRebel)



Java Enterprise

- Which versions of **Java Enterprise** do you use? (JetBrains, JakartaEE)



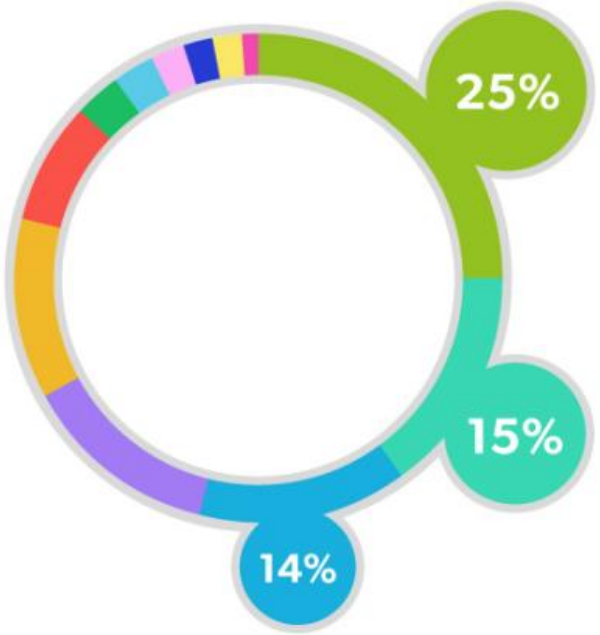


Cloud Platforms

- Top **Cloud Platform providers**? (Jakarta EE)

PaaS Providers

- If you use a platform, who is your **PaaS provider**? (JRebel)

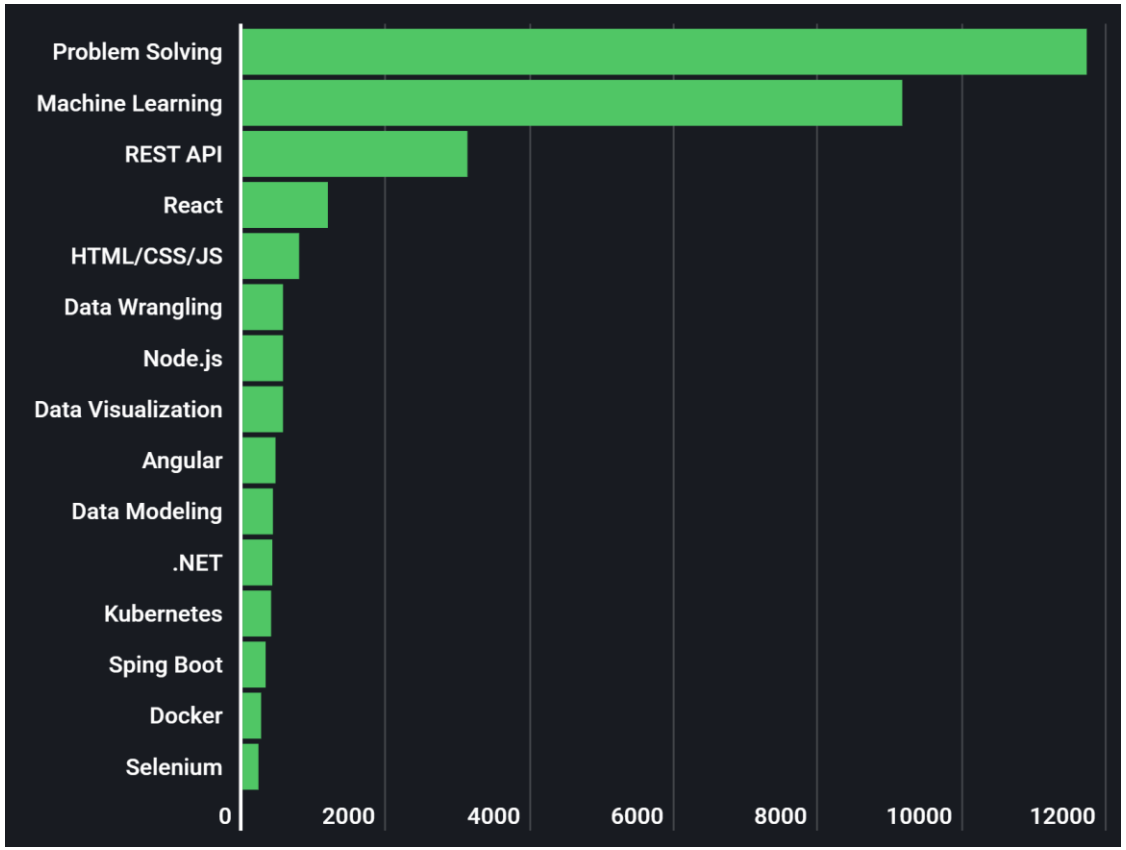


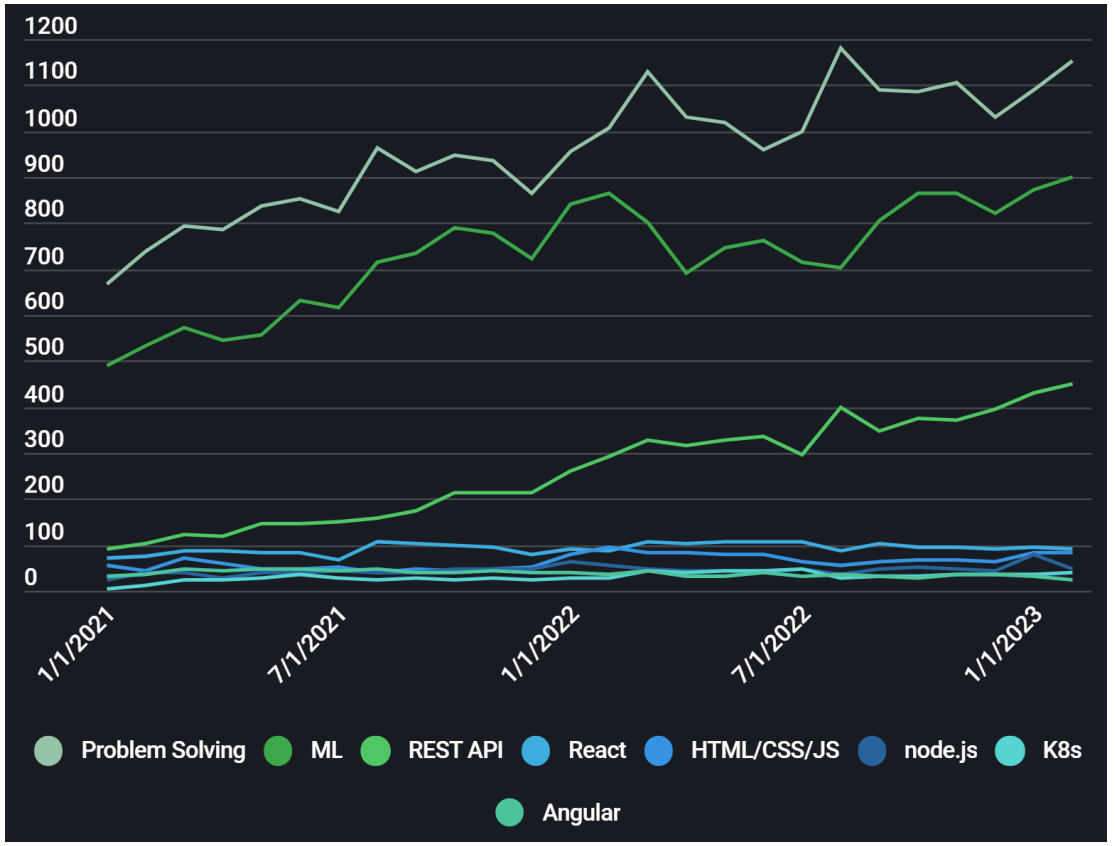
AWS	25%
Microsoft Azure	15%
We don't use PaaS providers	14%
IBM Cloud	13%
Red Hat Openshift	12%
Google Cloud Platform	8%
Oracle Cloud Platform	3%
Other	3%
VMWare Tanzu	2%
Alibaba Cloud	2%
SAP Cloud Platform	2%
Pivotal Cloud Foundry	1%

- How to learn?

Skills in Demand

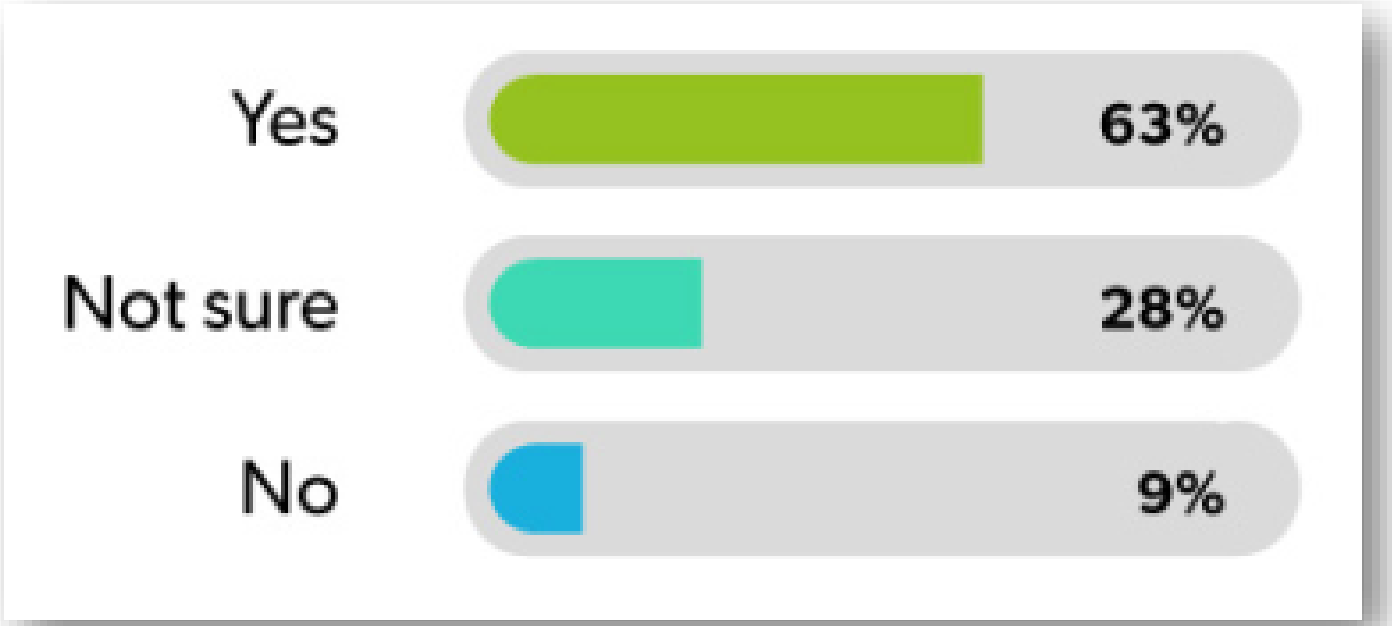
- **Skills in demand** by sum of monthly active mandatory tests? (HackerRank)





Hiring Developers

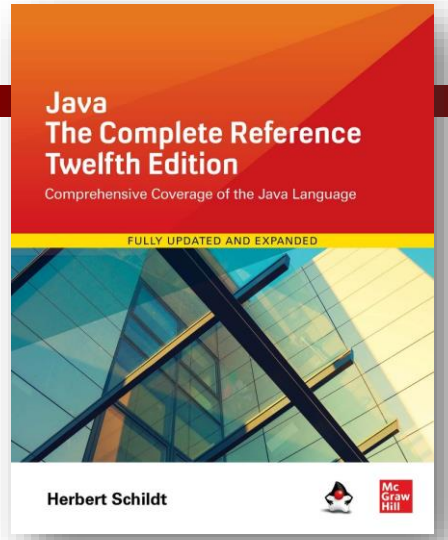
- Does your company have plans to **hire additional Java Developers** in 2023? (JRebel)?

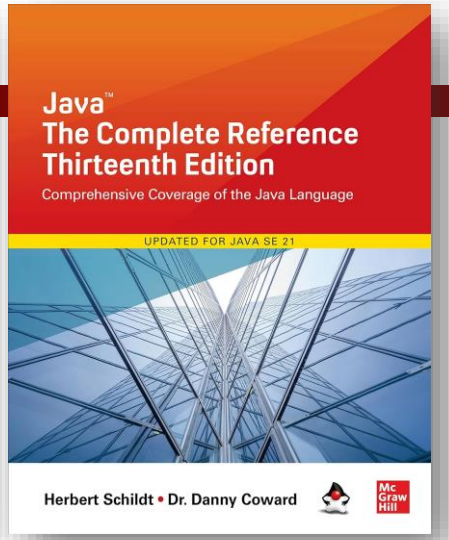


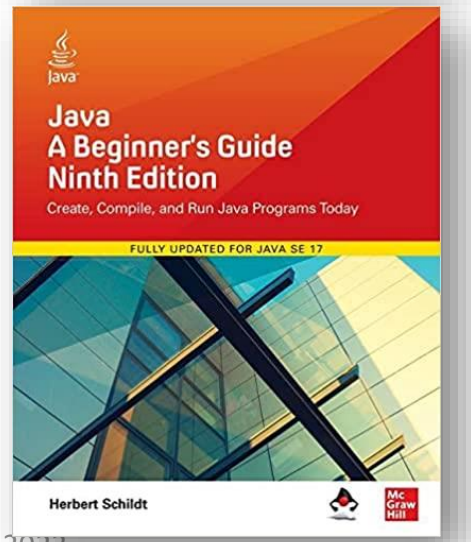
- Do they need to **know Java** first?

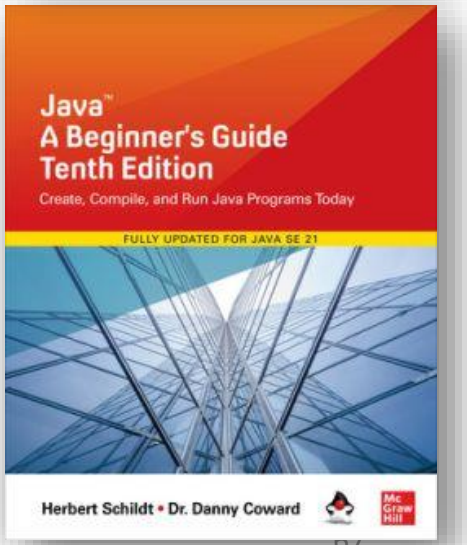
Java Books

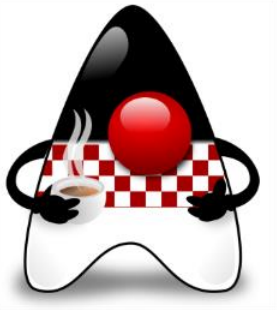
- **Java: The Complete Reference, 12th ed.**, Herbert Schildt, MGH, November 2021
  - Covering Java 17
  - 1280 pages, 32-39 €
- 13th edition covering Java 21 will be available in January 2024
- **Java: A Beginner's Guide, 9th ed.**, Herbert Schildt, MGH, January 2022
  - Covering Java 17
  - 728 pages, 32 €
- 10th edition covering Java 21 will be available in March 2024





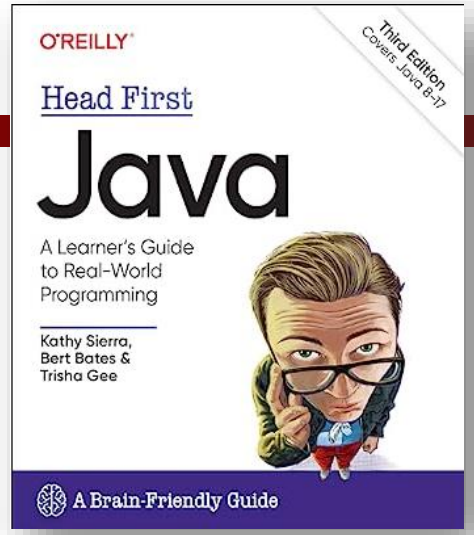


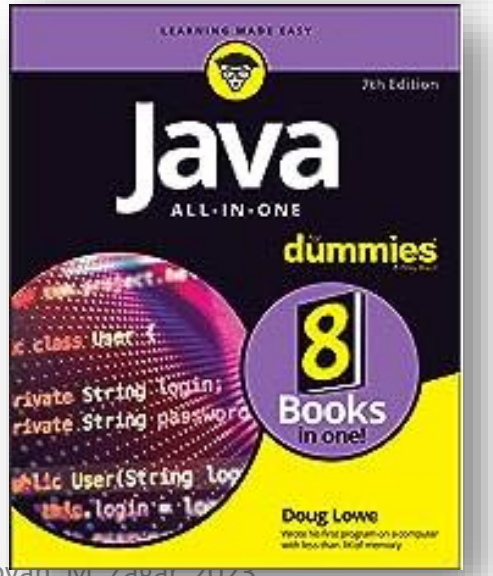


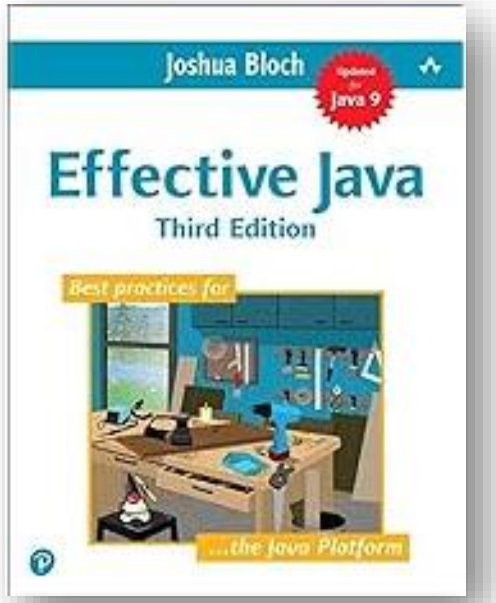


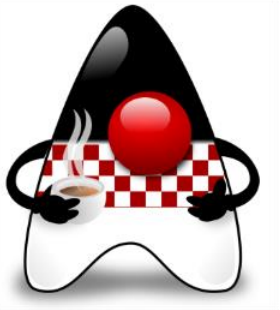
Java Books

- **Head First Java: A Brain-Friendly Guide, 3rd ed.**, Kathy Sierra, Bert Bates, Trisha Gee, O'Reilly, 2022
  - Covering Java 17
  - 4.7*, 752 pages, 56€
- **Java All-in-One For Dummies 7th ed.**, Doug Lowe, For Dummies, Feb 2023
  - Covering Java 19
  - 4.7*, 912 pages, 33$
- **Effective Java 3rd ed.**, Joshua Bloch, Pearson-AW, 2017
  - Older, 4.7*, 416 pages, 40$



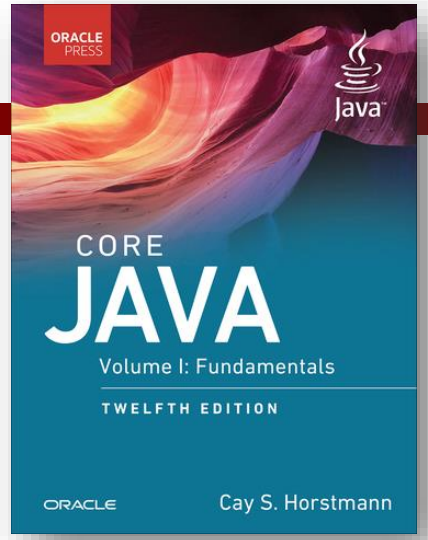




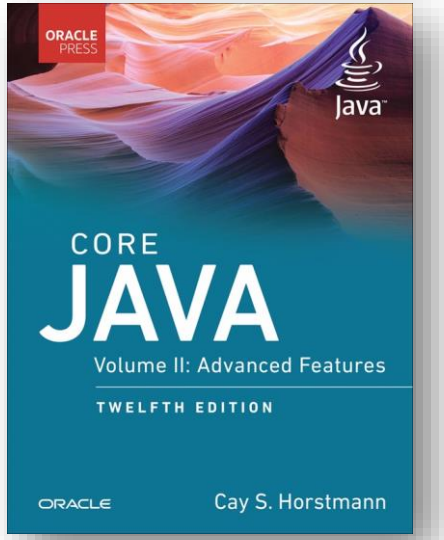


Java Books

- **Core Java, Volume I: Fundamentals, 12th ed.,** Cay S. Horstmann, December 2021, Oracle Press
  - Covering Java 17
  - 911 pages, approx. 60 €

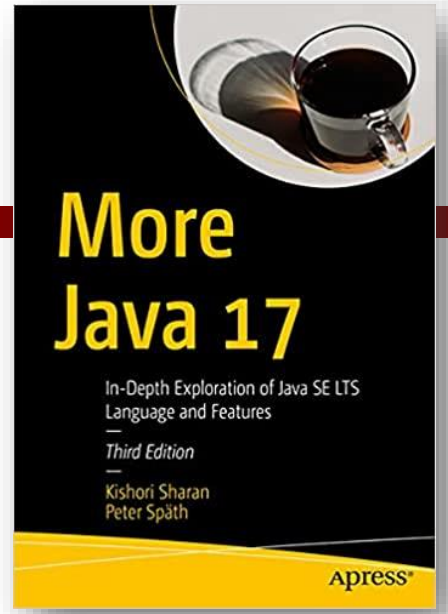


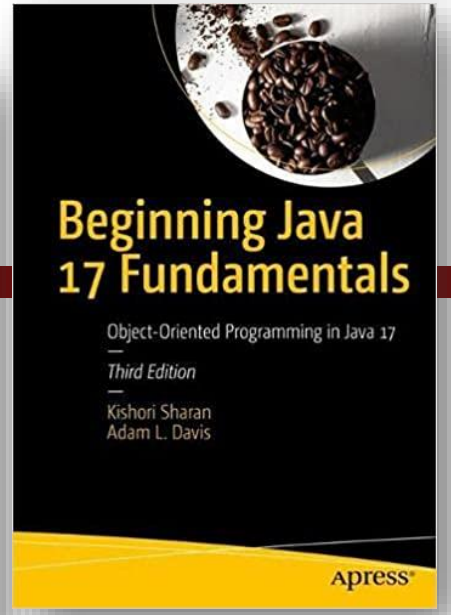
- **Core Java, Volume II: Advanced Features, 12th ed.,** Cay S. Horstmann, April 2022, Oracle Press
  - Covering Java 17
  - 944 pages, approx. 50 €

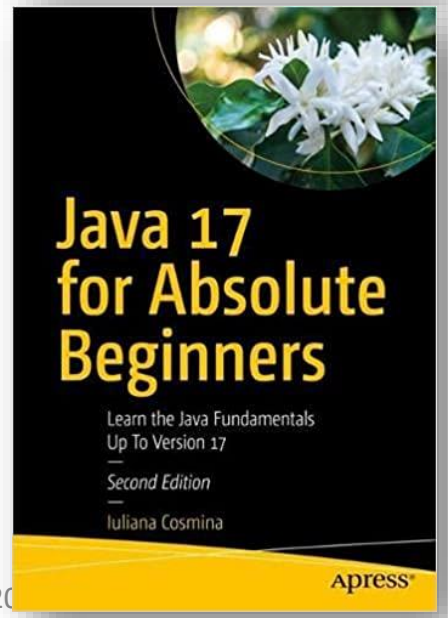


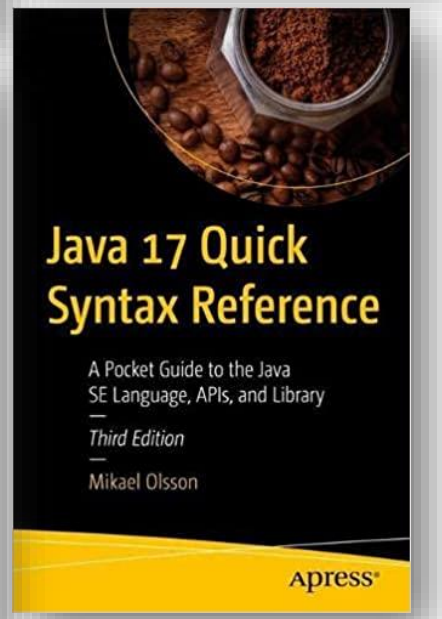
Java Books

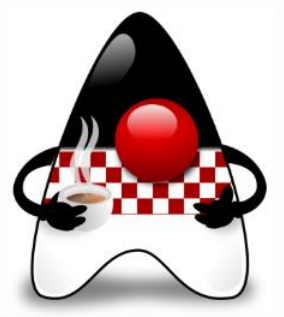
- **Beginning Java 17 Fundamentals, 3rd ed.** by Kishori Sharan, Adam L. Davis, Apress, Nov 2021
  - 9781484273067, 800 pages, approx. 45 €
- **More Java 17, 3rd ed.** by Kishori Sharan, Peter Späth, Apress, Dec 2021
  - 9781484271346, approx. 64 €
- **Java 17 Quick Syntax Reference** by Mikael Olsson, Apress, Oct 2021
  - 9781484273708, 218 pages, approx. 26 €
- **Java 17 for Absolute Beginners** by Iuliana Cosmina, Apress, Dec 2021
  - 9781484270790, 600 pages, approx. 42 €



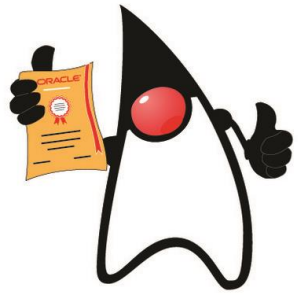




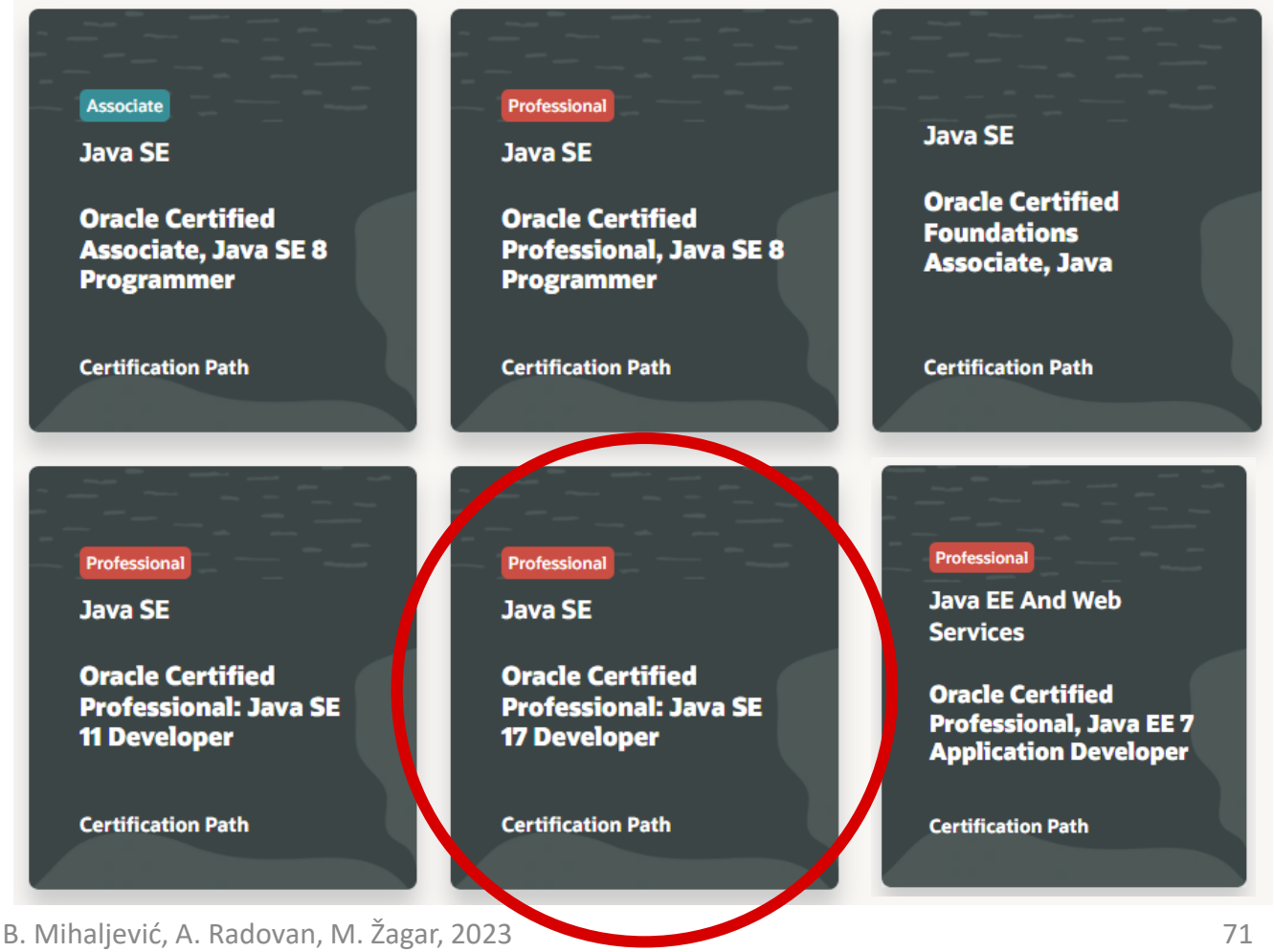




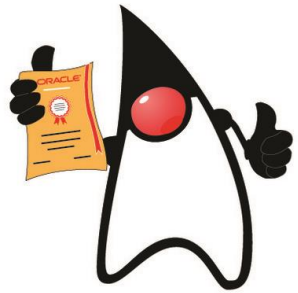
Java Certification



- **Oracle Certified Professional (OCP): Java SE 17 Developer**
  - Exam: Java SE 17 Developer **1Z0-829**
- There are also:
  - Oracle Certified Professional (OCP): Java SE 11 Developer
    - Exam: Java SE 11 Developer 1Z0-819
  - Oracle Certified Professional (OCP): Java SE 8 Programmer
  - Oracle Certified Professional (OCP): Helidon Microservices Developer
- Where is **Java 21** exam?



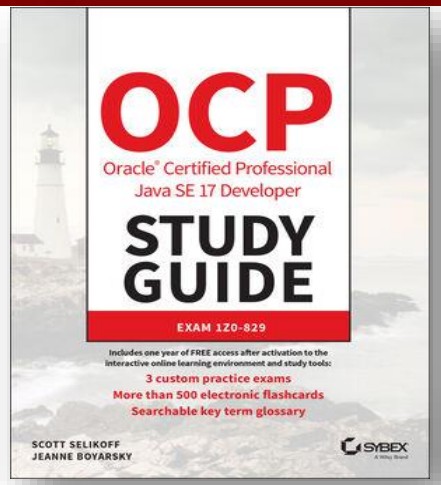
Java Certifications

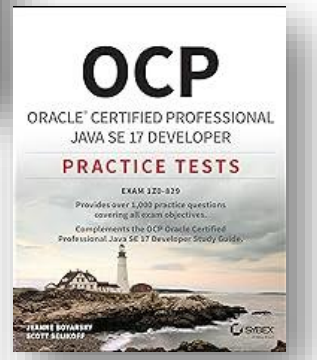


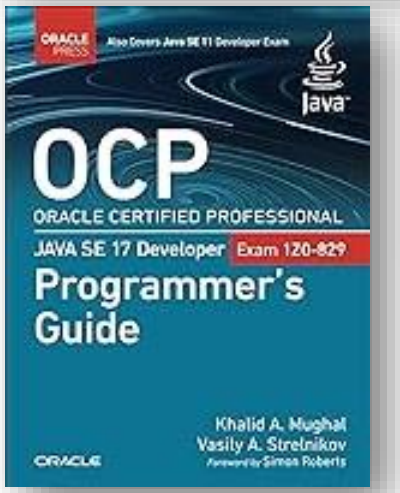
- **Oracle Certified Professional (OCP): Java SE 17 Developer**
  - Exam: Java SE 17 Developer 1Z0-829
  - Price: **228 €** | Duration: **90 Minutes** | Passing Score: **68%**

Study Guides:

- **OCP Oracle Certified Professional Java SE 17 Developer Study Guide: Exam 1Z0-829,** by Scott Selikoff and Jeanne Boyarsky, May 2022, Wiley
  - 1056 pages, Amazon rating: 4.7*, approx. 45 €
  - Additional practice tests book, approx. 32 €
- **OCP Oracle Certified Professional Java SE 17 Developer (Exam 1Z0-829) Programmer's Guide** by Mughal, Strelnikov and Roberts, Oracle Press
  - 1803 pages, Amazon rating: only 2.9-3.1*, approx. 77€







Is Java really progressing?

This is only our Java community opinion ☺

- **Frequent** Java releases **every 6 months** with many JEPS ✓
- **Java LTS** releases every **2 years** for production ✓
- Faster access **to new features** and **many improvement ideas** ✓
- Still, a lot of **maintenance and housekeeping** ✓
- Java is (finally) **free** and **going forward!** ✓

We are looking forward to new things!

Instead of the conclusion

Use Java 21 LTS ☺
or the previous Java 17 LTS
or (at least) Java 11 LTS

- **Any OpenJDK** – **it's up to you** ☺
- Try to **abandon older versions** (older than Java 17, and definitely Java 8 or older) and **double-check** what is **@Deprecated**
- If possible, **migrate** at every **2 years** with LTS
- **Get involved** more with **HUJAK** and visit more **conferences**!

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