Java 24
Горячие JEP’ы

Андрей Кулешов

Вадим Цесько

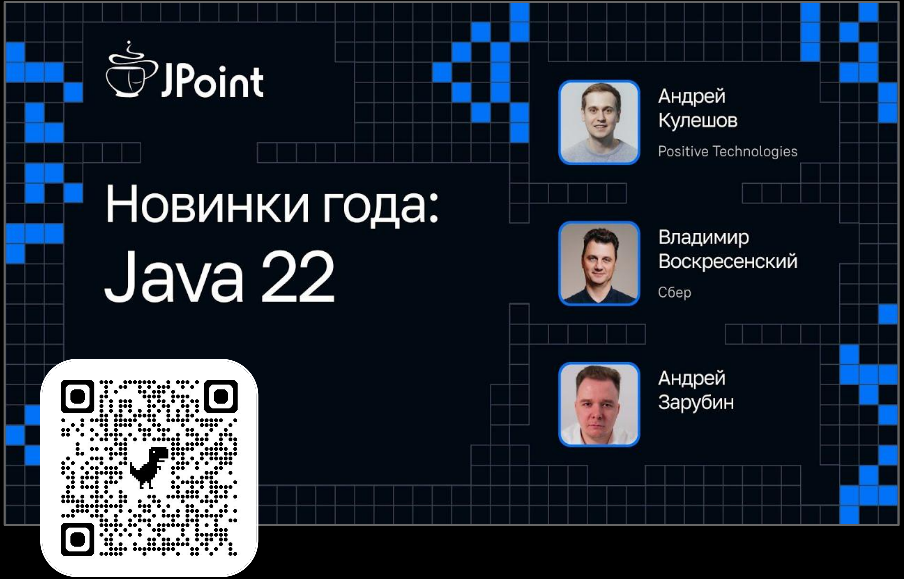
Владимир Ситников

О чем поговорим?

Oracle Java SE Support Roadmap*†

Release	GA Date	Premier Support Until	Extended Support Until
8 (LTS)**	March 2014	March 2022	December 2030*****
9 - 10 (non-LTS)	September 2017 - March 2018	March 2018 - September 2018	Not Available
11 (LTS)	September 2018	September 2023	January 2032*****
12 - 16 (non-LTS)	March 2019 - March 2021	September 2019 - September 2021	Not Available
17 (LTS)	September 2021	September 2026****	September 2029****
18 - 20 (non-LTS)	March 2022 - March 2023	September 2022 - September 2023	Not Available
21 (LTS)	September 2023	September 2028****	September 2031****
22 (non-LTS)	March 2024	September 2024	Not Available
23 (non-LTS)***	September 2024	March 2025	Not Available
24 (non-LTS)***	March 2025	September 2025	Not Available
25 (LTS)***	September 2025	September 2030	September 2033

В предыдущих сериях



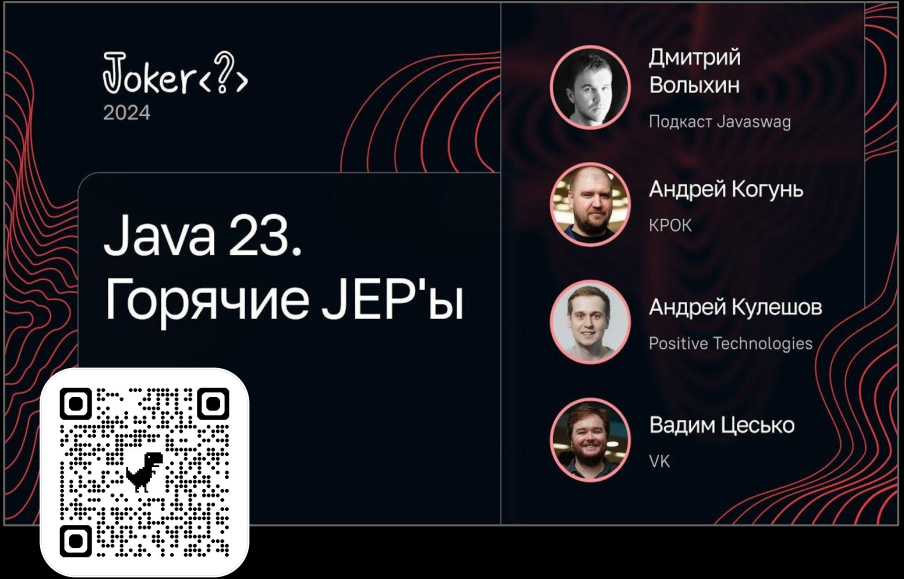
JPoint

Новинки года:
Java 22

Андрей Кулешов
Positive Technologies

Владимир Воскресенский
Сбер

Андрей Зарубин



Joker<?>
2024

Java 23.
Горячие JEP'ы

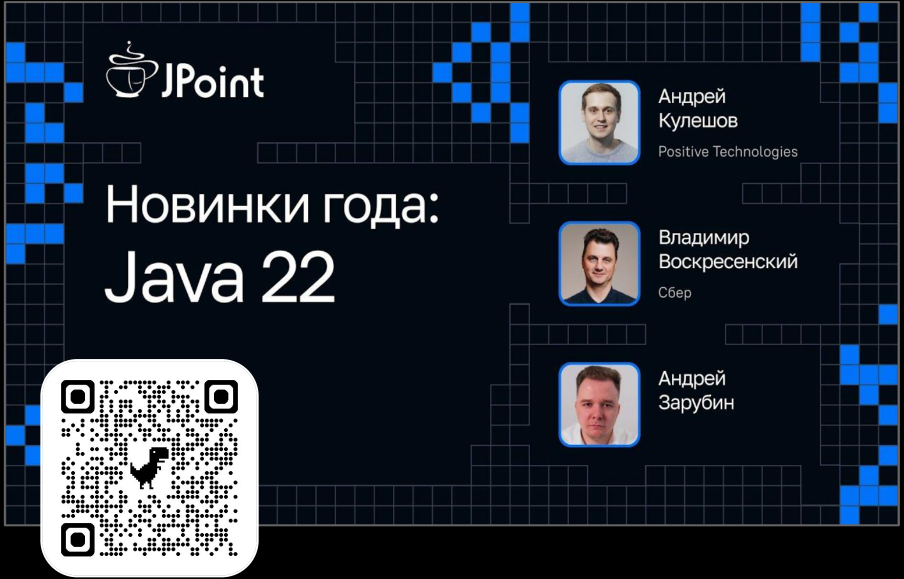
Дмитрий Волыхин
Подкаст Javaswag

Андрей Когунь
КРОК

Андрей Кулешов
Positive Technologies

Вадим Цесько
VK

В предыдущих сериях

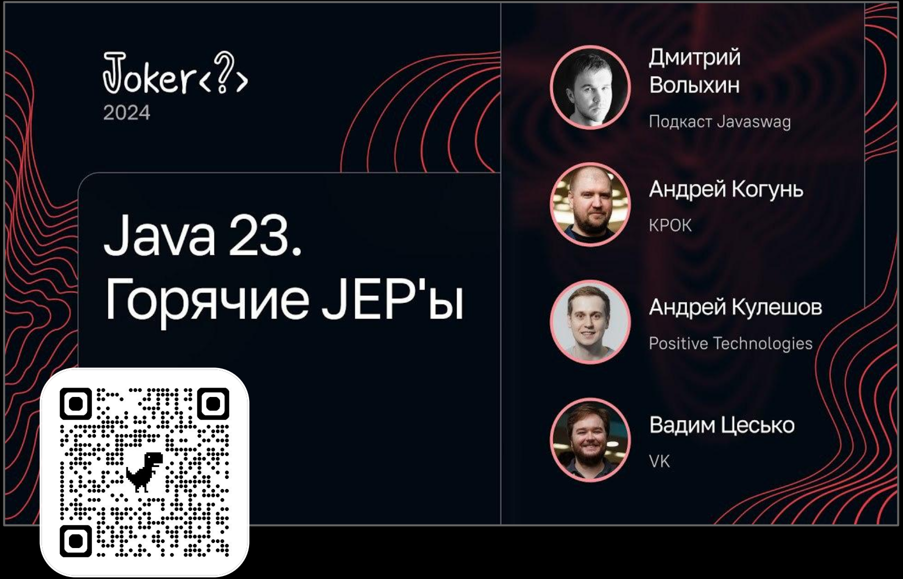

JPoint

Новинки года: Java 22

Андрей Кулешов
Positive Technologies

Владимир Воскресенский
Сбер

Андрей Зарубин


Joker<?> 2024

Java 23. Горячие JEP'ы

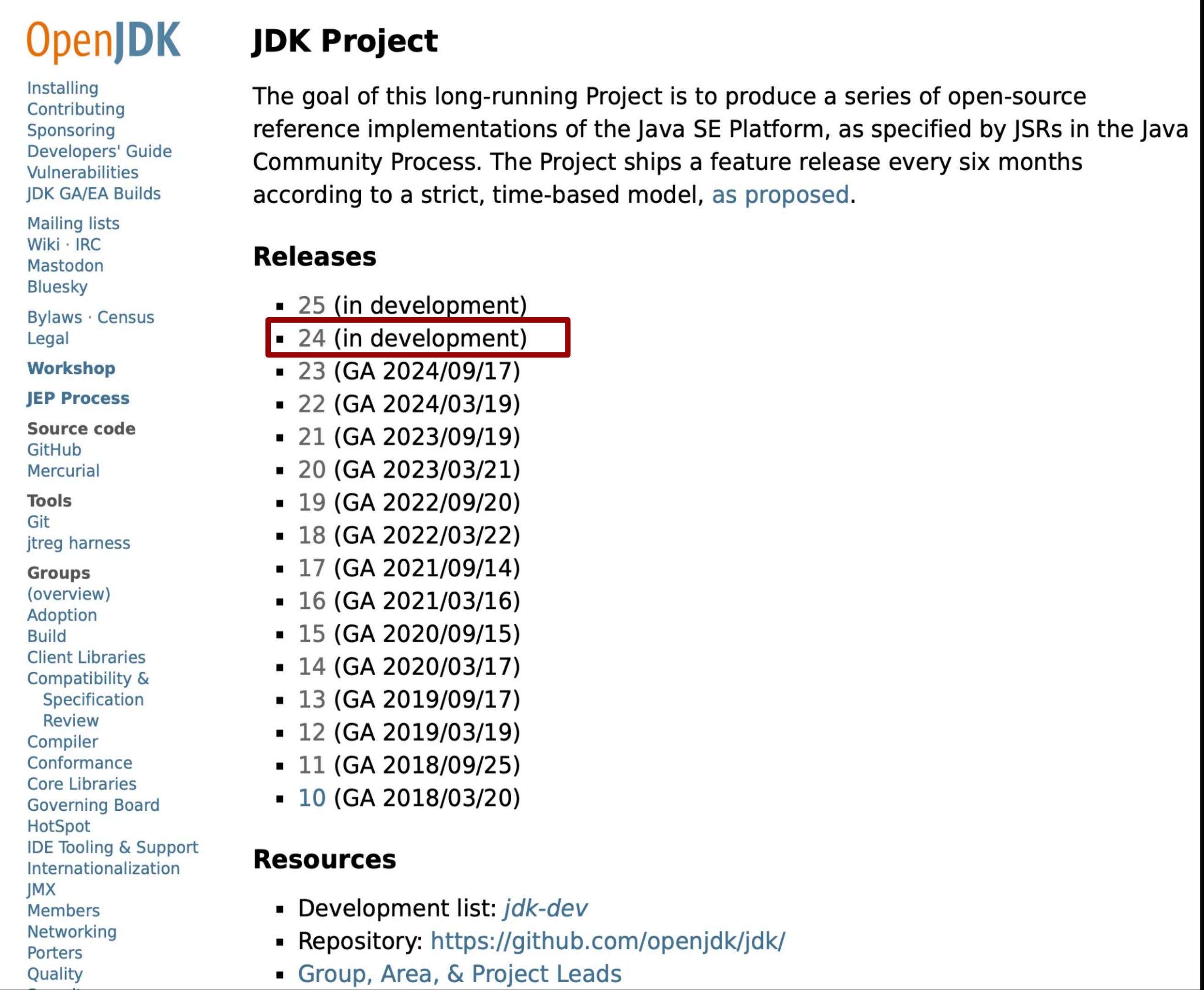
Дмитрий Волыхин
Подкаст Javaswag

Андрей Когунь
КРОК

Андрей Кулешов
Positive Technologies

Вадим Цесько
VK

Большая благодарность Диме и Андрею
за помощь в подготовке!



OpenJDK

Installing
Contributing
Sponsoring
Developers' Guide
Vulnerabilities
JDK GA/EA Builds

Mailing lists
Wiki · IRC
Mastodon
Bluesky

Bylaws · Census
Legal

Workshop

JEP Process

Source code
GitHub
Mercurial

Tools
Git
jtreg harness

Groups
(overview)
Adoption
Build
Client Libraries
Compatibility & Specification Review
Compiler
Conformance
Core Libraries
Governing Board
HotSpot
IDE Tooling & Support
Internationalization
JMX
Members
Networking
Porters
Quality

JDK Project

The goal of this long-running Project is to produce a series of open-source reference implementations of the Java SE Platform, as specified by JSRs in the Java Community Process. The Project ships a feature release every six months according to a strict, time-based model, as proposed.

Releases

- 25 (in development)
- $\boxed{\text{24 (in development)}}$
- 23 (GA 2024/09/17)
- 22 (GA 2024/03/19)
- 21 (GA 2023/09/19)
- 20 (GA 2023/03/21)
- 19 (GA 2022/09/20)
- 18 (GA 2022/03/22)
- 17 (GA 2021/09/14)
- 16 (GA 2021/03/16)
- 15 (GA 2020/09/15)
- 14 (GA 2020/03/17)
- 13 (GA 2019/09/17)
- 12 (GA 2019/03/19)
- 11 (GA 2018/09/25)
- 10 (GA 2018/03/20)

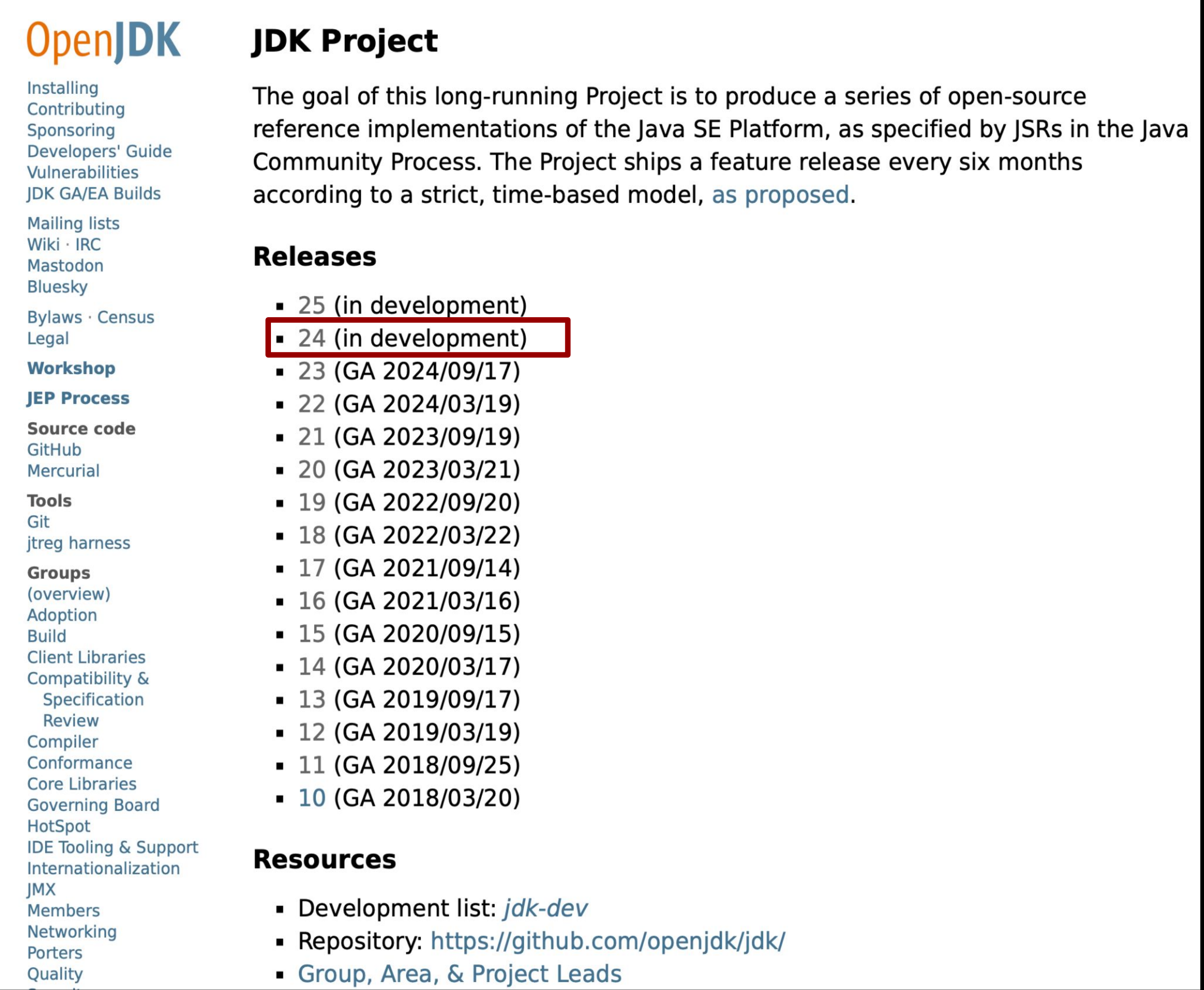
Resources

- Development list: *jdk-dev*
- Repository: https://github.com/openjdk/jdk/
- Group, Area, & Project Leads

Initial Release
Candidate 2025/02/06

Final Release
Candidate 2025/02/20

General Availability
2025/03/18


OpenJDK

Installing
Contributing
Sponsoring
Developers' Guide
Vulnerabilities
JDK GA/EA Builds

Mailing lists
Wiki · IRC
Mastodon
Bluesky

Bylaws · Census
Legal

Workshop

JEP Process

Source code
GitHub
Mercurial

Tools
Git
jtreg harness

Groups
(overview)
Adoption
Build
Client Libraries
Compatibility & Specification Review
Compiler
Conformance
Core Libraries
Governing Board
HotSpot
IDE Tooling & Support
Internationalization
JMX
Members
Networking
Porters
Quality

JDK Project

The goal of this long-running Project is to produce a series of open-source reference implementations of the Java SE Platform, as specified by JSRs in the Java Community Process. The Project ships a feature release every six months according to a strict, time-based model, as proposed.

Releases

- 25 (in development)
- 24 (in development)
- 23 (GA 2024/09/17)
- 22 (GA 2024/03/19)
- 21 (GA 2023/09/19)
- 20 (GA 2023/03/21)
- 19 (GA 2022/09/20)
- 18 (GA 2022/03/22)
- 17 (GA 2021/09/14)
- 16 (GA 2021/03/16)
- 15 (GA 2020/09/15)
- 14 (GA 2020/03/17)
- 13 (GA 2019/09/17)
- 12 (GA 2019/03/19)
- 11 (GA 2018/09/25)
- 10 (GA 2018/03/20)

Resources

- Development list: *jdk-dev*
- Repository: https://github.com/openjdk/jdk/
- Group, Area, & Project Leads

Initial Release
Candidate 2025/02/06

Final Release
Candidate 2025/02/20

General Availability
2025/03/18

КТО УЖЕ ПЕРЕШЁЛ?

Что вас зацепило?

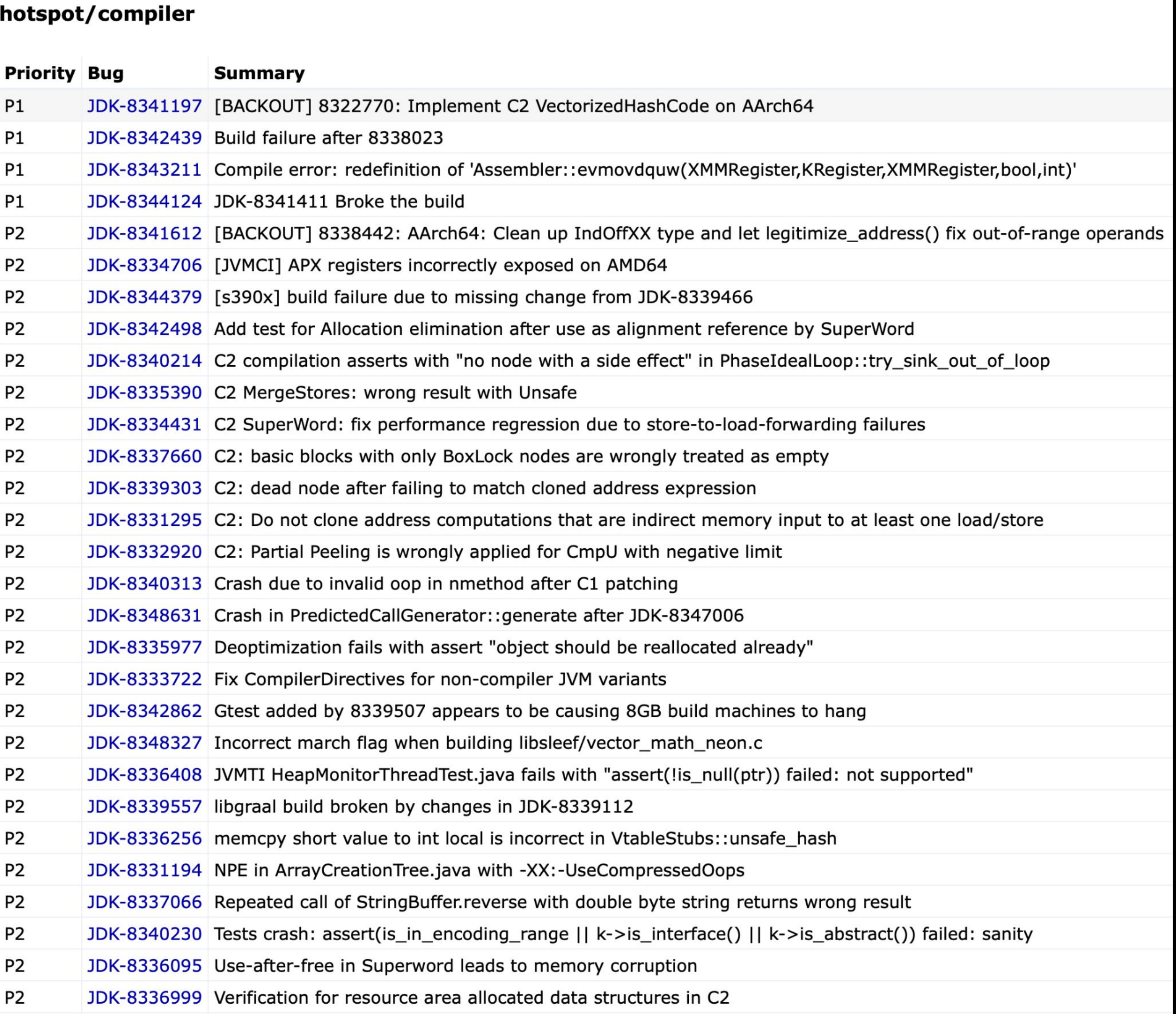
В цифрах (наши подсчеты)

24 JEP’a

15 new features (не JEP)

4 known issues

6 notable fixes



hotspot/compiler

Priority	Bug	Summary
P1	JDK-8341197	[BACKOUT] 8322770: Implement C2 VectorizedHashCode on AArch64
P1	JDK-8342439	Build failure after 8338023
P1	JDK-8343211	Compile error: redefinition of 'Assembler::evmovdquw(XMMRegister,KRegister,XMMRegister,bool,int)'
P1	JDK-8344124	JDK-8341411 Broke the build
P2	JDK-8341612	[BACKOUT] 8338442: AArch64: Clean up IndOffXX type and let legitimize_address() fix out-of-range operands
P2	JDK-8334706	[JVMCI] APX registers incorrectly exposed on AMD64
P2	JDK-8344379	[s390x] build failure due to missing change from JDK-8339466
P2	JDK-8342498	Add test for Allocation elimination after use as alignment reference by SuperWord
P2	JDK-8340214	C2 compilation asserts with "no node with a side effect" in PhaseIdealLoop::try_sink_out_of_loop
P2	JDK-8335390	C2 MergeStores: wrong result with Unsafe
P2	JDK-8334431	C2 SuperWord: fix performance regression due to store-to-load-forwarding failures
P2	JDK-8337660	C2: basic blocks with only BoxLock nodes are wrongly treated as empty
P2	JDK-8339303	C2: dead node after failing to match cloned address expression
P2	JDK-8331295	C2: Do not clone address computations that are indirect memory input to at least one load/store
P2	JDK-8332920	C2: Partial Peeling is wrongly applied for CmpU with negative limit
P2	JDK-8340313	Crash due to invalid oop in nmethod after C1 patching
P2	JDK-8348631	Crash in PredictedCallGenerator::generate after JDK-8347006
P2	JDK-8335977	Deoptimization fails with assert "object should be reallocated already"
P2	JDK-8333722	Fix CompilerDirectives for non-compiler JVM variants
P2	JDK-8342862	Gtest added by 8339507 appears to be causing 8GB build machines to hang
P2	JDK-8348327	Incorrect march flag when building libsleef/vector_math_neon.c
P2	JDK-8336408	JVMTI HeapMonitorThreadTest.java fails with "assert(!is_null(ptr)) failed: not supported"
P2	JDK-8339557	libgraal build broken by changes in JDK-8339112
P2	JDK-8336256	memcpy short value to int local is incorrect in VtableStubs::unsafe_hash
P2	JDK-8331194	NPE in ArrayCreationTree.java with -XX:-UseCompressedOops
P2	JDK-8337066	Repeated call of StringBuffer.reverse with double byte string returns wrong result
P2	JDK-8340230	Tests crash: assert(is_in_encoding_range || k->is_interface() || k->is_abstract()) failed: sanity
P2	JDK-8336095	Use-after-free in Superword leads to memory corruption
P2	JDK-8336999	Verification for resource area allocated data structures in C2

2700+

Но мы будем говорить про JEPы

Шапочно: стабильные JEPы

484: Class-File API
485: Stream Gatherers
491: Synchronize Virtual Threads without Pinning
483: Ahead-of-Time Class Loading & Linking

$\Bigg\}$ пользовательские

472: Prepare to Restrict the Use of JNI
498: Warn upon Use of Memory-Access Methods in Unsafe

501: Deprecate the 32-bit x86 Port for Removal
479: Remove the Windows 32-bit x86 Port
486: Permanently Disable the Security Manager
490: ZGC: Remove the Non-Generational Mode

493: Linking Run-Time Images without JMODs
475: Late Barrier Expansion for G1

496: Quantum-Resistant Module-Lattice-Based Key Encapsulation Mechanism
497: Quantum-Resistant Module-Lattice-Based Digital Signature Algorithm

Шапочно: стабильные JEPы

484: Class-File API
485: Stream Gatherers
491: Synchronize Virtual Threads without Pinning
483: Ahead-of-Time Class Loading & Linking

472: Prepare to Restrict the Use of JNI
498: Warn upon Use of Memory-Access Methods in Unsafe

$\left.\begin{array}{l}\\ \\ \end{array}\right\}$ нотификации

501: Deprecate the 32-bit x86 Port for Removal
479: Remove the Windows 32-bit x86 Port
486: Permanently Disable the Security Manager
490: ZGC: Remove the Non-Generational Mode

493: Linking Run-Time Images without JMODs
475: Late Barrier Expansion for G1

496: Quantum-Resistant Module-Lattice-Based Key Encapsulation Mechanism
497: Quantum-Resistant Module-Lattice-Based Digital Signature Algorithm

Шапочно: стабильные JEPы

484: Class-File API
485: Stream Gatherers
491: Synchronize Virtual Threads without Pinning
483: Ahead-of-Time Class Loading & Linking

472: Prepare to Restrict the Use of JNI
498: Warn upon Use of Memory-Access Methods in Unsafe

$$\left.\begin{array}{l}\text{501: Deprecate the 32-bit x86 Port for Removal}\\ \text{479: Remove the Windows 32-bit x86 Port}\\ \text{486: Permanently Disable the Security Manager}\\ \text{490: ZGC: Remove the Non-Generational Mode}\end{array}\right\}\text{удаление старого}$$

493: Linking Run-Time Images without JMODs
475: Late Barrier Expansion for G1

496: Quantum-Resistant Module-Lattice-Based Key Encapsulation Mechanism
497: Quantum-Resistant Module-Lattice-Based Digital Signature Algorithm

Шапочно: стабильные JEPы

484: Class-File API
485: Stream Gatherers
491: Synchronize Virtual Threads without Pinning
483: Ahead-of-Time Class Loading & Linking

472: Prepare to Restrict the Use of JNI
498: Warn upon Use of Memory-Access Methods in Unsafe

501: Deprecate the 32-bit x86 Port for Removal
479: Remove the Windows 32-bit x86 Port
486: Permanently Disable the Security Manager
490: ZGC: Remove the Non-Generational Mode

$\left.\begin{array}{l}\text{493: Linking Run-Time Images without JMODs}\\ \text{475: Late Barrier Expansion for G1}\end{array}\right\}$ технические вещи

496: Quantum-Resistant Module-Lattice-Based Key Encapsulation Mechanism
497: Quantum-Resistant Module-Lattice-Based Digital Signature Algorithm

Шапочно: стабильные JEPы

484: Class-File API
485: Stream Gatherers
491: Synchronize Virtual Threads without Pinning
483: Ahead-of-Time Class Loading & Linking

472: Prepare to Restrict the Use of JNI
498: Warn upon Use of Memory-Access Methods in Unsafe

501: Deprecate the 32-bit x86 Port for Removal
479: Remove the Windows 32-bit x86 Port
486: Permanently Disable the Security Manager
490: ZGC: Remove the Non-Generational Mode

493: Linking Run-Time Images without JMODs
475: Late Barrier Expansion for G1

$$\left.\begin{array}{l}\text{496: Quantum-Resistant Module-Lattice-Based Key Encapsulation Mechanism}\\ \text{497: Quantum-Resistant Module-Lattice-Based Digital Signature Algorithm}\end{array}\right\} Q$$

Шапочно: preview

478: Key Derivation Function API (Preview)

494: Module Import Declarations (Second Preview)
488: Primitive Types in Patterns, instanceof, and switch (Second Preview)

492: Flexible Constructor Bodies (Third Preview)

487: Scoped Values (Fourth Preview)
495: Simple Source Files and Instance Main Methods (Fourth Preview)
499: Structured Concurrency (Fourth Preview)

Шапочно: incubator + experimental

404: Generational Shenandoah (Experimental)

450: Compact Object Headers (Experimental)

489: Vector API (Ninth Incubator)

485: Stream Gatherers

485: Stream Gatherers

- Как было раньше

```
stream.map(...).filter(...).map(...).collect(...)
```

485: Stream Gatherers

- Теперь можно делать свои промежуточные операции

```
stream.map(...).gather(...).gather(...).collect(...)
```

485: Stream Gatherers

- Теперь можно делать свои промежуточные операции

  ```
  stream.map(...).gather(...).gather(...).collect(...)
  ```

-
  ```
  Stream.of(1,2,3,4,5,6,7,8,9).gather(new WindowFixed(3)).toList()
  ⟹ [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
  ```

485: Stream Gatherers

```java
record WindowFixed<T>(int size) implements Gatherer<T, ArrayList<T>, List<T>> {
      ...

   @Override public Supplier<ArrayList<T>> initializer() {
      ...
   }

   @Override public Integrator<ArrayList<T>, T, List<T>> integrator() {
       ...
   }

   @Override public BiConsumer<ArrayList<T>, Downstream<? super List<T >>> finisher() {
      ...
   }
}
```

485: Stream Gatherers

- Новые методы ищем в классе Gatherers

```
Gatherers.windowFixed
Gatherers.windowSliding
Gatherers.fold
Gatherers.scan
Gatherers.mapConcurrent
```

485: Stream Gatherers

- Новые методы ищем в классе Gatherers

```
Gatherers.windowFixed
Gatherers.windowSliding
Gatherers.fold
Gatherers.scan
Gatherers.mapConcurrent vs .parallel().map() ?
```

491: Synchronize Virtual Threads without Pinning

491: Synchronize Virtual Threads without Pinning

- Как было: виртуальные потоки **не могли “отвязаться”** (unmount) **от потока-носителя** (carrier) в synchronized блоках

```
synchronized byte[] getData() {
    byte[] buf = ...;
    // Может заблокироваться здесь, читая байты из сокета
    int nread = socket.getInputStream().read(buf);
...
}
```

491: Synchronize Virtual Threads without Pinning

- Как было: synchronized сводило на нет фичи виртуальных потоков

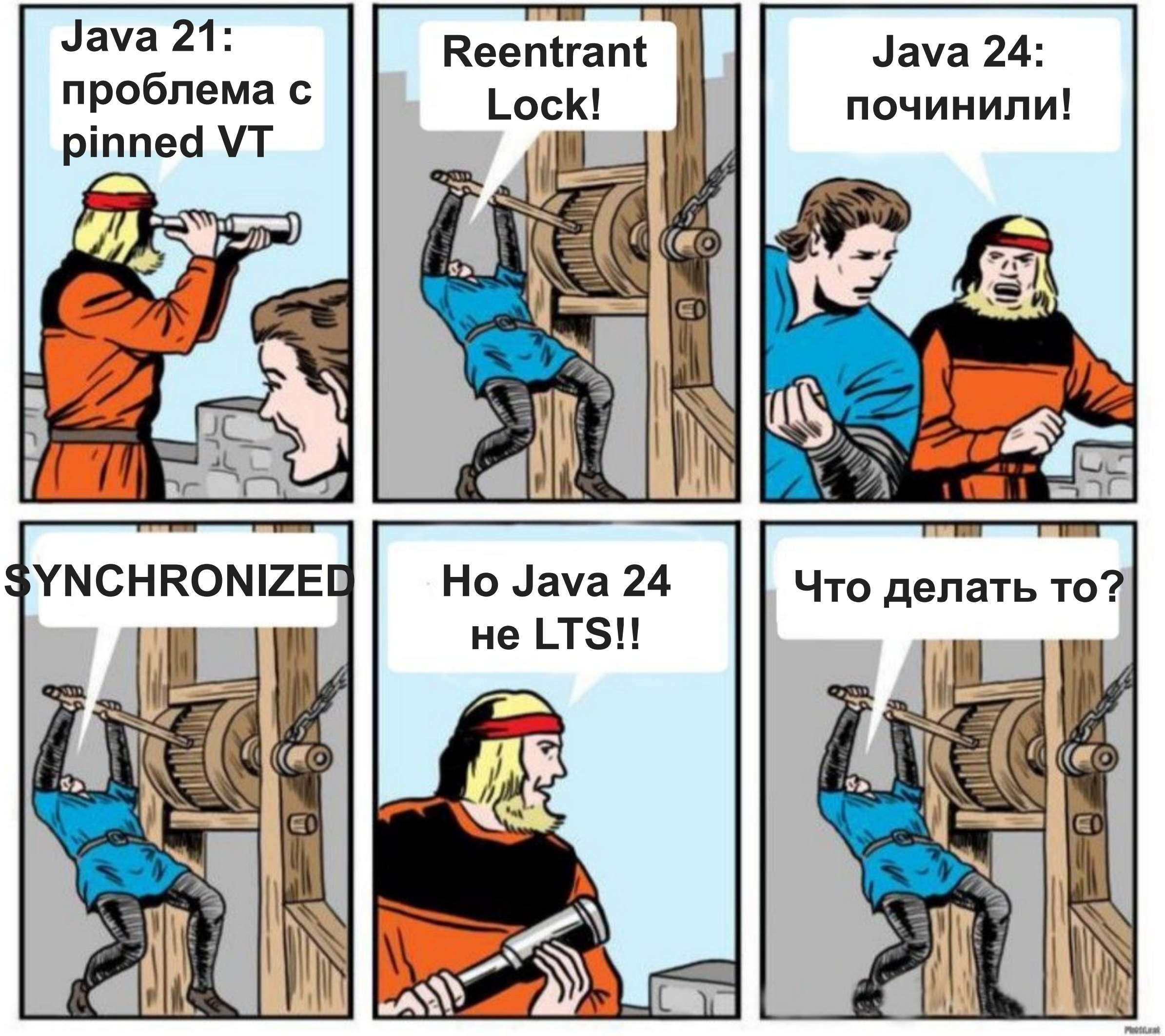
```
synchronized byte[] getData() {
    byte[] buf = ...;
    // Может заблокироваться здесь, читая байты из сокета
    int nread = socket.getInputStream().read(buf);
...
}
```

491: Synchronize Virtual Threads without Pinning

- Как стало: виртуальные потоки могут переключаться даже при использовании synchronized*

- synchronized не будет жестко привязывать потоки-носители, они будут свободны для других задач

- Миграция на ReentrantLock больше не требуется


Java 21: проблема с pinned VT
Reentrant Lock!
Java 24: починили!
SYNCHRONIZED
Но Java 24 не LTS!!
Что делать то?

483: Ahead-of-Time Class Loading & Linking

483: Ahead-of-Time Class Loading & Linking

Запуск JVM приложения — это:

- Сканирование и парсинг тысяч класс-файлов
- Линковка классов, проверку байт-кода, резолв ссылок
- Статические инициализаторы, создание объектов и т.д.

Улучшаем время запуска: классы мгновенно доступны, загружены и залинкованы при старте HotSpot.

483: Ahead-of-Time Class Loading & Linking

- Запускаем ~~гуся, работяги~~, собираем статистику и конфигурацию AOT:

```
java -XX:AOTMode=record -XX:AOTConfiguration=app.aotconf -cp app.jar com.example.App
```

483: Ahead-of-Time Class Loading & Linking

- Запускаем ~~гуся, работяги~~, собираем статистику и конфигурацию AOT:

  ```
  java -XX:AOTMode=record -XX:AOTConfiguration=app.aotconf -cp app.jar com.example.App
  ```

- Создаём AOT кэш с информацией о линковке:

  ```
  java -XX:AOTMode=create -XX:AOTConfiguration=app.aotconf -XX:AOTCache=app.aot -cp app.jar
  ```

483: Ahead-of-Time Class Loading & Linking

- Запускаем ~~гуся, работяги~~, собираем статистику и конфигурацию AOT:

```
java -XX:AOTMode=record -XX:AOTConfiguration=app.aotconf -cp app.jar com.example.App
```

- Создаём AOT кэш с информацией о линковке:

```
java -XX:AOTMode=create -XX:AOTConfiguration=app.aotconf -XX:AOTCache=app.aot -cp app.jar
```

- Запускаем приложение с AOT кэшом:

```
java -XX:AOTCache=app.aot -cp app.jar com.example.App
```

483: Ahead-of-Time Class Loading & Linking

- Запускаем ~~гуся, работяги~~, собираем статистику и конфигурацию AOT:

  ```
  java -XX:AOTMode=record -XX:AOTConfiguration=app.aotconf -cp app.jar com.example.App
  ```

- Создаём AOT кэш с информацией о линковке:

  ```
  java -XX:AOTMode=create -XX:AOTConfiguration=app.aotconf -XX:AOTCache=app.aot -cp app.jar
  ```

- Запускаем приложение с AOT кэшом:

  ```
  java -XX:AOTCache=app.aot -cp app.jar com.example.App
  ```

- “If cache is unavailable or incompatible, JVM issues a warning and continues.”

483: Ahead-of-Time Class Loading & Linking

Spring PetClinic (21,000 classes):

Startup time reduced from

4.48s → 2.60s (42% improvement)

483: Ahead-of-Time Class Loading & Linking

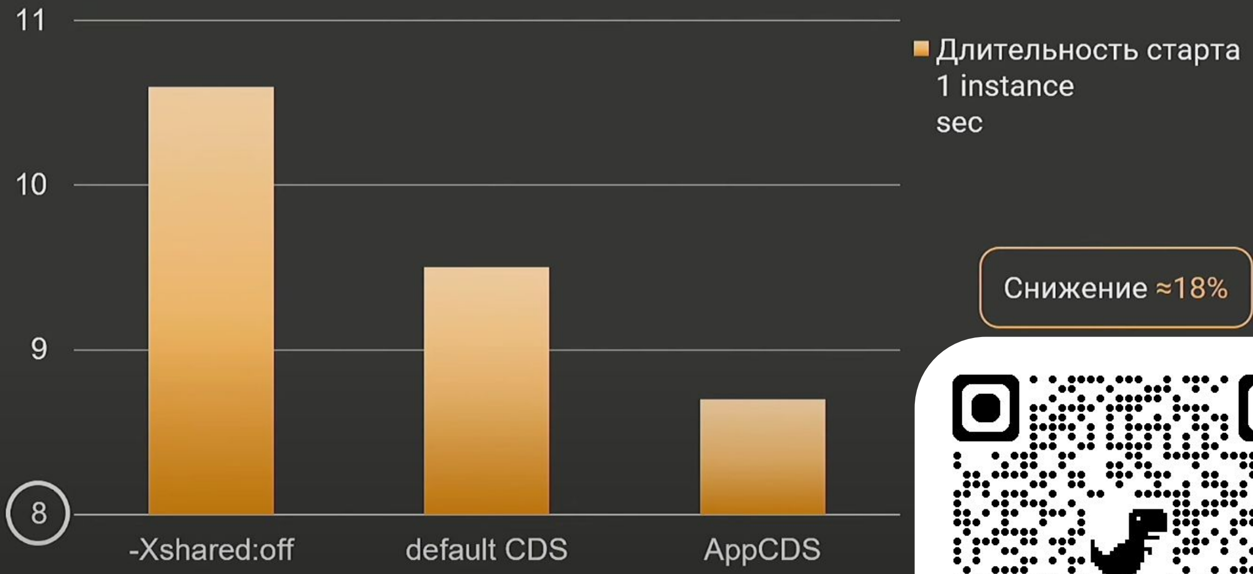
Spring PetClinic (21,000 classes):

Startup time reduced from

4.48s → 2.60s (42% improvement) Java 24

4.48s → 3.00s (33% improvement) Java 13+, AppCDS

Время запуска микросервиса



Снижение ≈18%

484: Class-File API

484: Class-File API

- Замена ASM и прочих библиотек для работы с байткодом вроде ByteBuddy/Soot/etc
- Устранение взаимной зависимости JDK и ASM

Зарелизьте ASM
и мы сможем выкатить OpenJDK
Зарелизьте OpenJDK
и мы сможем выкатить
ASM

484: Class-File API: что хотим

```
String getConference(boolean spring) {
    if (spring) {
        return "JPoint";
    } else {
        return "Joker";
    }
}
```

484: Class-File API: как генерируем

```
ClassFile.of()
 .build(ClassDesc.of("ConferenceService"), classBuilder →
   classBuilder.withMethod(
     "getConference",
     MethodTypeDesc.of(CD_String),
     AccessFlag.PUBLIC.mask(),
     methodBuilder → methodBuilder.withFlags(AccessFlag.STATIC)
       .withCode(codeBuilder → codeBuilder.iload(codeBuilder.parameterSlot(0))
           .ifThenElse(
             th → th.ldc(classBuilder.constantPool().stringEntry("JPoint"))
                     .return_(),
             el → el.ldc(classBuilder.constantPool().stringEntry("Joker"))
                     .return_()
           )
       )
   )
 );
```

478: Key Derivation Function API (Preview)

478: Key Derivation Function API (Preview)

- KDF: Функция формирования ключа по секрету
- Поддержка алгоритмов HKDF (RFC 5869) и Argon2 (RFC 9106)
- Шаг в сторону пост-квантовой безопасности

478: Key Derivation Function API (Preview)

```
KDF hkdf = KDF.getInstance("HKDF-SHA256");

AlgorithmParameterSpec params =
      HKDFParameterSpec.ofExtract()
          .addIKM(initialKeyMaterial)
          .addSalt(salt)
          .thenExpand(info, 32);

SecretKey key = hkdf.deriveKey("AES", params);
```

478: Key Derivation Function API (Preview)

```
KDF hkdf = KDF.getInstance("HKDF-SHA256");

AlgorithmParameterSpec params =
          HKDFParameterSpec.ofExtract()
                  .addIKM(initialKeyMaterial)
                  .addSalt(salt).thenExpand(info, 32);

SecretKey key = hkdf.deriveKey("AES", params);
```

“A KDF implementation must extend the abstract class `javax.crypto.KDFSpi`”

478: Key Derivation Function API (Preview)

```
@PreviewFeature(feature = PreviewFeature.Feature.KEY_DERIVATION)
public abstract class KDFSpi {

    protected abstract KDFParameters engineGetParameters();

    protected abstract SecretKey engineDeriveKey(...);

    protected abstract byte[] engineDeriveData(...)

}
```

494: Module Import Declarations (Second Preview)

494: Module Import Declarations (Second Preview)

```
import module M1;
```

494: Module Import Declarations (Second Preview)

```
import module M1;


import module java.base;
import module java.se;
```

494: Module Import Declarations (Second Preview)

```
import module M1;


import module java.base;
import module java.se;
```

Тот самый легендарный пакет с пакетами!

494: Module Import Declarations (Second Preview)

```java
// exports java.util, which has a public Date class
import module java.base;
// exports java.sql, which has a public Date class
import module java.sql;


public static void main(String[] args) {
   Date d = new Date(...); // ???
}
```

494: Module Import Declarations (Second Preview)

```java
// exports java.util, which has a public Date class
import module java.base;
// exports java.sql, which has a public Date class
import module java.sql;

import java.sql.Date;


public static void main(String[] args) {
    Date d = new Date(...); // ???
}
```

494: Module Import Declarations (Second Preview)

```java
// exports java.util, which has a public Date class
import module java.base;
// exports java.sql, which has a public Date class
import module java.sql;

import java.sql.Date;


public static void main(String[] args) {
    Date d = new Date(...); // ???
}
```

494: Module Import Declarations (Second Preview)

Что нового в сравнении с Java 23?

- java.se теперь включает и классы java.base
- type-import-on-demand перекрывают импорты модулей

```
import module java.base;
import module java.sql;

import java.sql.*;


public static void main(String[] args) {
    Date d = new Date(...); // ???
}
```

488: Primitive Types in Patterns, instanceof, and switch

(Second Preview)

488: Primitive Types in Patterns, instanceof, and switch (Second Preview)

Primitive types in
instanceof and switch

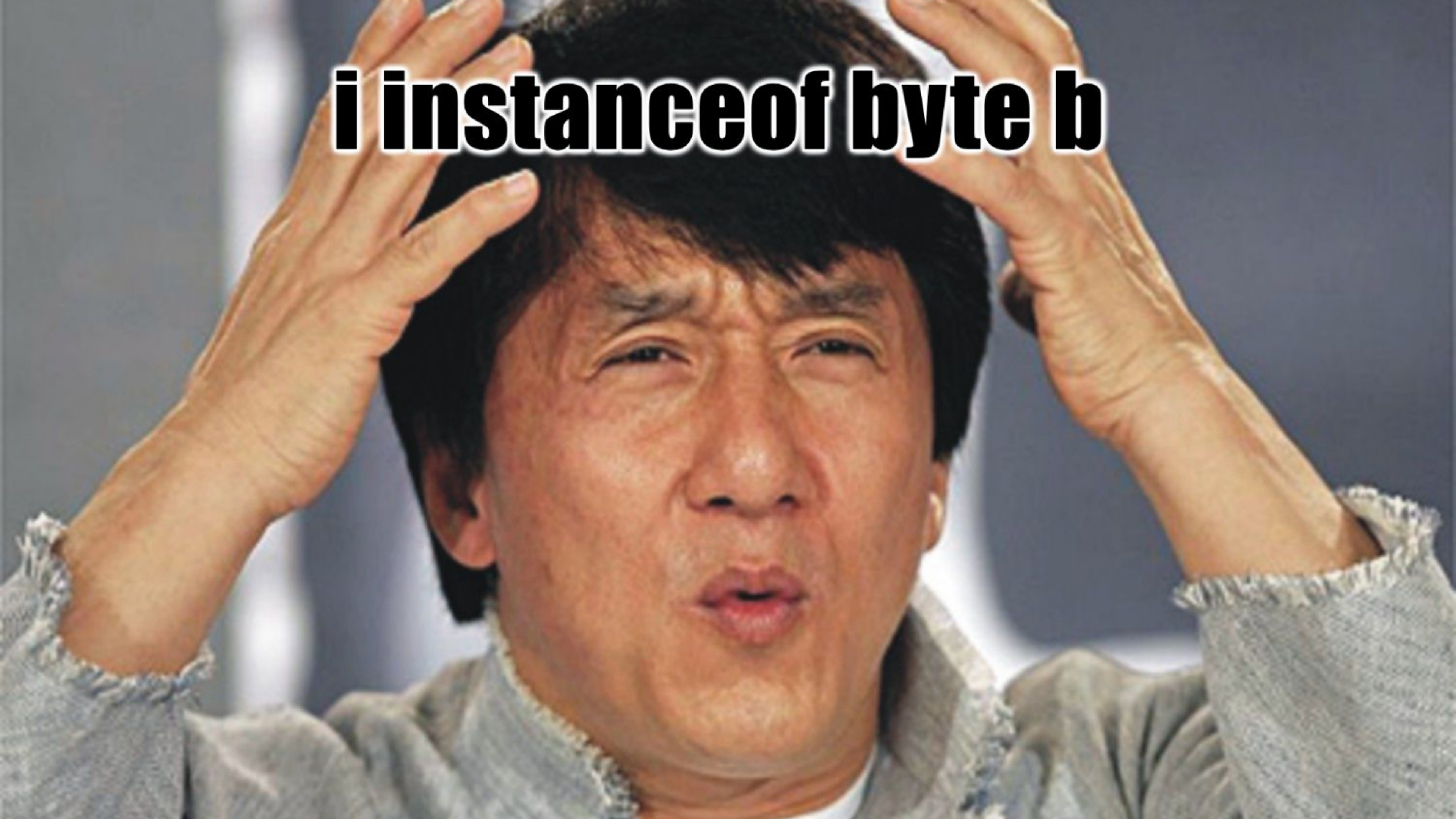
488: Primitive Types in Patterns, instanceof, and switch (Second Preview)

```java
var i = 15;
if (i >= -128 && i <= 127) {
   byte b = (byte) i;
}
```

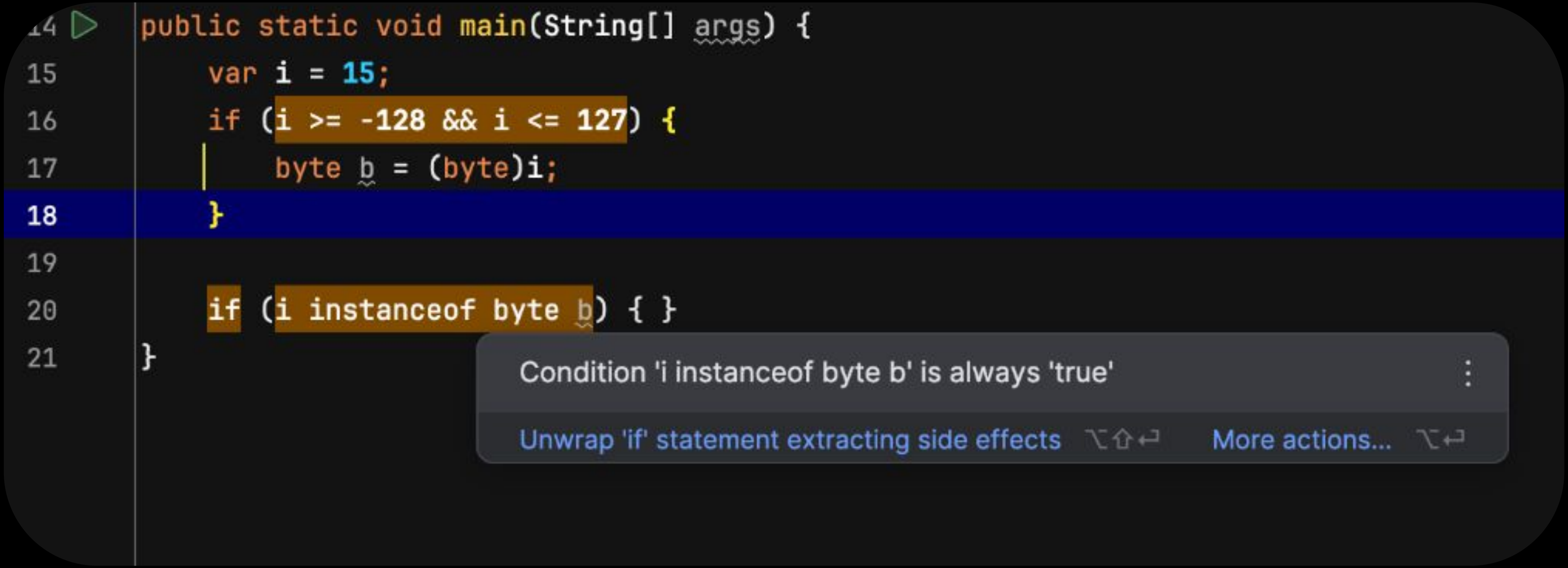
488: Primitive Types in Patterns, instanceof, and switch (Second Preview)

```
var i = 15;
if (i ⩾ -128 && i ⩽ 127) {
   byte b = (byte) i;
}

// теперь можно так:
if (i instanceof byte b) { }
```


i instanceof byte b

488: Primitive Types in Patterns, instanceof, and switch (Second Preview)



```java
public static void main(String[] args) {
    var i = 15;
    if (i >= -128 && i <= 127) {
        byte b = (byte)i;
    }

    if (i instanceof byte b) { }
}
```

Condition 'i instanceof byte b' is always 'true'

488: Primitive Types in Patterns, instanceof, and switch (Second Preview)

БЕЗ изменений!

“We here propose to preview it for a second time, without change.”

488: Primitive Types in Patterns, instanceof, and switch (Second Preview)

Pattern matching for switch **does not** support primitive type patterns

488: Primitive Types in Patterns, instanceof, and switch (Second Preview)

```
var a = switch (x.getStatus()) {
    case 0 → "okay";
    case 1 → "warning";
    case 2 → "error";
    case int i → "unknown status: " + i;
};
```

488: Primitive Types in Patterns, instanceof, and switch (Second Preview)

Record patterns have **limited** support for primitive types

```java
sealed interface JsonValue {
   record JsonString(String s) implements JsonValue { }
   record JsonNumber(double d) implements JsonValue { }
   record JsonObject(Map<String, JsonValue> map) implements JsonValue { }
}

var json = new JsonObject(Map.of(
         "name", new JsonString("John"),
         "age", new JsonNumber(30))
);

if (json instanceof JsonObject(var map) &&
      map.get("age") instanceof JsonNumber(double a)) {
         int age = (int)a;
}
```

```java
sealed interface JsonValue {
   record JsonString(String s) implements JsonValue { }
   record JsonNumber(double d) implements JsonValue { }
   record JsonObject(Map<String, JsonValue> map) implements JsonValue { }
}

var json = new JsonObject(Map.of(
         "name", new JsonString("John"),
         "age", new JsonNumber(30))
);

if (json instanceof JsonObject(var map) &&
      map.get("age") instanceof JsonNumber(double a)) {
         int age = (int) a;
}
```

```java
sealed interface JsonValue {
   record JsonString(String s) implements JsonValue { }
   record JsonNumber(double d) implements JsonValue { }
   record JsonObject(Map<String, JsonValue> map) implements JsonValue { }
}

var json = new JsonObject(Map.of(
         "name", new JsonString("John"),
         "age", new JsonNumber(30))
);

if (json instanceof JsonObject(var map) &&
     map.get("age") instanceof JsonNumber(double a)) {
         int age = (int) a;
}
```

```java
sealed interface JsonValue {
    record JsonString(String s) implements JsonValue { }
    record JsonNumber(double d) implements JsonValue { }
    record JsonObject(Map<String, JsonValue> map) implements JsonValue { }
}

var json = new JsonObject(Map.of(
        "name", new JsonString("John"),
        "age", new JsonNumber(30))
);

if (json instanceof JsonObject(var map) &&
      map.get("age") instanceof JsonNumber(double a)) {
        int age = (int) a;
}
```

```
sealed interface JsonValue {
   record JsonString(String s) implements JsonValue { }
   record JsonNumber(double d) implements JsonValue { }
   record JsonObject(Map<String, JsonValue> map) implements JsonValue { }
}

var json = new JsonObject(Map.of(
         "name", new JsonString("John"),
         "age", new JsonNumber(30))
);

if (json instanceof JsonObject(var map) &&
      map.get("age") instanceof JsonNumber(int age)) {

}
```

492: Flexible Constructor Bodies (Third Preview)

492: Flexible Constructor Bodies (Third Preview)

Раньше приходилось валидировать так:

```java
public class PositiveBigInteger extends BigInteger {
    public PositiveBigInteger(long value) {
        super(verifyPositive(value));
    }

    private static long verifyPositive(long value) {
        if (value ⩽ 0)
            throw new IllegalArgumentException("non-positive value");
        return value;
    }
}
```

492: Flexible Constructor Bodies (Third Preview)

Теперь можно прямо в конструкторе:

```
public class PositiveBigInteger extends BigInteger {
    public PositiveBigInteger(long value) {
        if (value ≤ 0) throw new IllegalArgumentException(..);
        super(value);
    }
}
```

492: Flexible Constructor Bodies (Third Preview)

“We here propose to preview it for a third time, without significant change.”

499: Structured Concurrency (Fourth Preview)

499: Structured Concurrency (Fourth Preview)

We here propose to re-preview the API once more in JDK 24, without change, to give more time for feedback from real world usage.

Но мы напомним

```
public class StructuredConcurrencyExample {
   public void handle() throws ExecutionException, InterruptedException {
       try (var scope = new StructuredTaskScope.ShutdownOnFailure()) {

           Supplier<String> user = scope.fork(this::findUser);
           Supplier<Integer> id = scope.fork(this::fetchOrder);
           scope.join().throwIfFailed();

           println(user.get())
           println(id.get)
       }
   }

   private String findName() {
       return "User";
   }

   private Integer findId() {
       return 42;
   }
}
```

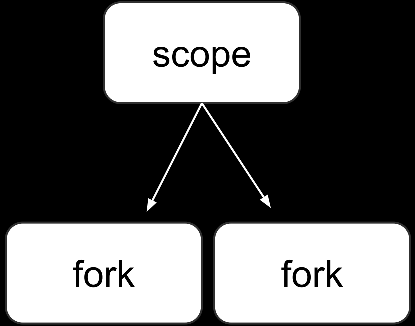
```java
public class StructuredConcurrencyExample {
   public void handle() throws ExecutionException, InterruptedException {
       try (var scope = new StructuredTaskScope.ShutdownOnFailure()) {

           Supplier<String> user = scope.fork(this::findUser);
           Supplier<Integer> id = scope.fork(this::fetchOrder);
           scope.join().throwIfFailed();

           println(user.get())
           println(id.get)
       }
   }

   private String findName() {
       return "User";
   }

   private Integer findId() {
       return 42;
   }
}
```



487: Scoped Values (Fourth Preview)

487: Scoped Values (Fourth Preview)

Задача

- Передать имя пользователя во все слои приложения
- Но пробрасывать его явно не хотим
- Можно задействовать ThreadLocal, но это боль

487: Scoped Values (Fourth Preview)

Проблемы с ThreadLocal

- Неограниченная мутабельность
- Неограниченный срок жизни и проблемы безопасности: `threadLocal.remove()`
- Дорогое наследование между потоками

Не вписываются в концепт Virtual Threads:

“However, if each of a million virtual threads has its own copy of thread-local variables, the memory footprint may be significant. In summary, thread-local variables have more complexity than is usually needed for sharing data”

487: Scoped Values (Fourth Preview)

```
class Framework {
    private static final ScopedValue<String> USERNAME = ScopedValue.newInstance();
```

487: Scoped Values (Fourth Preview)

```java
class Framework {
    private static final ScopedValue<String> USERNAME = ScopedValue.newInstance();

    void serve(Request request, Response response) {
        var username = getUsername(request);
        ScopedValue.where(USERNAME, username)
                .run(() → handle(request, response));
    }
```

487: Scoped Values (Fourth Preview)

```
class Framework {
    private static final ScopedValue<String> USERNAME = ScopedValue.newInstance();

    void serve(Request request, Response response) {
        var username = getUsername(request);
        ScopedValue.where(USERNAME, username)
                .run(() → handle(request, response));
    }

    void handle(Request request, Response response) {
        log.info("Username is: {}", USERNAME.get());
        ...
    }
```

487: Scoped Values (Fourth Preview)

Кто знает, что такое Fluent Interface?

487: Scoped Values (Fourth Preview)

"We removed the callWhere and runWhere methods from the ScopedValue class, leaving the API completely **fluent**."

487: Scoped Values (Fourth Preview)

“We removed the callWhere and runWhere methods from the ScopedValue class, leaving the API completely **fluent**.”

```
public static <T> void runWhere(ScopedValue<T> key, T value, Runnable op) {
    where(key, value).run(op);
}

public static <...> R callWhere(...) throws X {
    return where(key, value).call(op);
}
```

495: Simple Source Files and Instance Main Methods (Fourth Preview)

495: Simple Source Files and Instance Main Methods (Fourth Preview)

```
class HelloWorld {
    void main() {
        System.out.println("Hello, World!");
    }
}
```

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

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

495: Simple Source Files and Instance Main Methods (Fourth Preview)

```
class HelloWorld {
    void main() {
        System.out.println("Hello, World!");
    }
}
```

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

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

+ Automatic import of the java.base module

495: Simple Source Files and Instance Main Methods (Fourth Preview)

```
String greeting() { return "Hello, World!"; }

void main() {
    println(greeting());
}
```

495: Simple Source Files and Instance Main Methods (Fourth Preview)

```
final class FileName {
   ...
   main()V
    L0
     LINENUMBER 4 L0
     ALOAD 0
     INVOKEVIRTUAL FileName.greeting ()Ljava/lang/String;
     INVOKESTATIC java/io/IO.println (Ljava/lang/Object;)V
   ...
}
```

495: Simple Source Files and Instance Main Methods (Fourth Preview)

“We here propose to preview it for a fourth time, with **new terminology** and **a revised title** but otherwise unchanged, in order to gain additional experience and feedback.”

495: Simple Source Files and Instance Main Methods (Fourth Preview)

JEP 477: Implicitly Declared Classes and Instance Main Methods (Third Preview)

JEP 495: Simple Source Files and Instance Main Methods (Fourth Preview)

Что ещё?

А еще квантовые алгоритмы...😱

496: Quantum-Resistant Module-Lattice-Based Key Encapsulation Mechanism

497: Quantum-Resistant Module-Lattice-Based Digital Signature Algorithm

495: Quantum resistant: чего мы боимся?

- Текущая криптография держится на сложности разложения чисел на множители
- Квантовые компьютеры уже доступны
- Квантовые компьютеры раскладывают числа на множители на раз

495: Quantum resistant: на практике

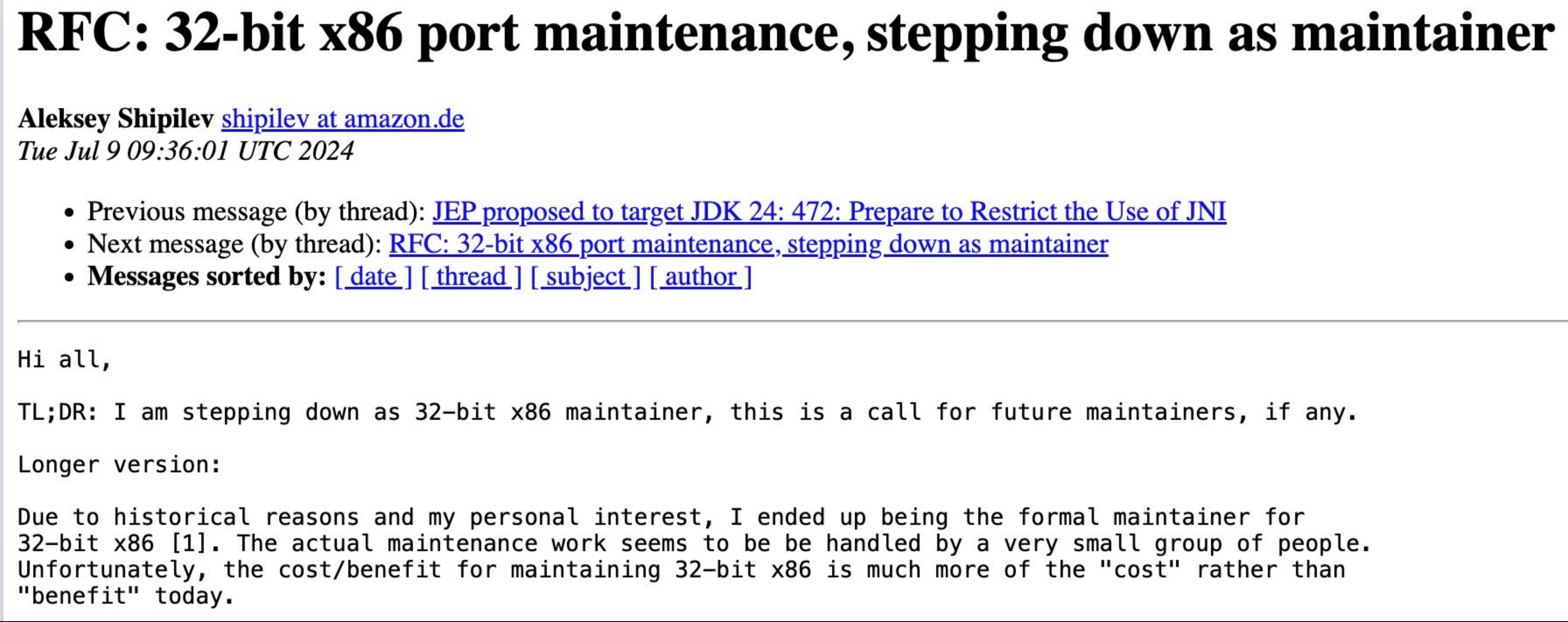
- в 2001 квантовый комп смог разложить 15 на прост
- в 2012 тоже смогли разложить 15 и потом 21
- в 2016 снова смогли разложить 15
- в 2019 попытались разложить 35, но не смогли из-за накапливающихся ошибок
- в 2022 китайцы разложили 261’980’999’226’229 (48 бит)
- Но китайский подход не масштабируется, поэтому простые множители пока в безопасности

JEP 498

```
WARNING: A terminally deprecated method in sun.misc.Unsafe has been called
WARNING: sun.misc.Unsafe::setMemory has been called by com.foo.bar.Server
WARNING: Please consider reporting this to the maintainers of com.foo.bar.Server
WARNING: sun.misc.Unsafe::setMemory will be removed in a future release
```

JEP 472

```
WARNING: A restricted method in java.lang.System has been called
WARNING: System::load has been called by com.foo.Server in module com.foo
WARNING: Use --enable-native-access=com.foo to avoid a warning for callers in this module
WARNING: Restricted methods will be blocked in a future release unless native access is enabled
```



RFC: 32-bit x86 port maintenance, stepping down as maintainer

Aleksey Shipilev shipilev at amazon.de
Tue Jul 9 09:36:01 UTC 2024

- Previous message (by thread): JEP proposed to target JDK 24: 472: Prepare to Restrict the Use of JNI
- Next message (by thread): RFC: 32-bit x86 port maintenance, stepping down as maintainer
- **Messages sorted by:** [date] [thread] [subject] [author]

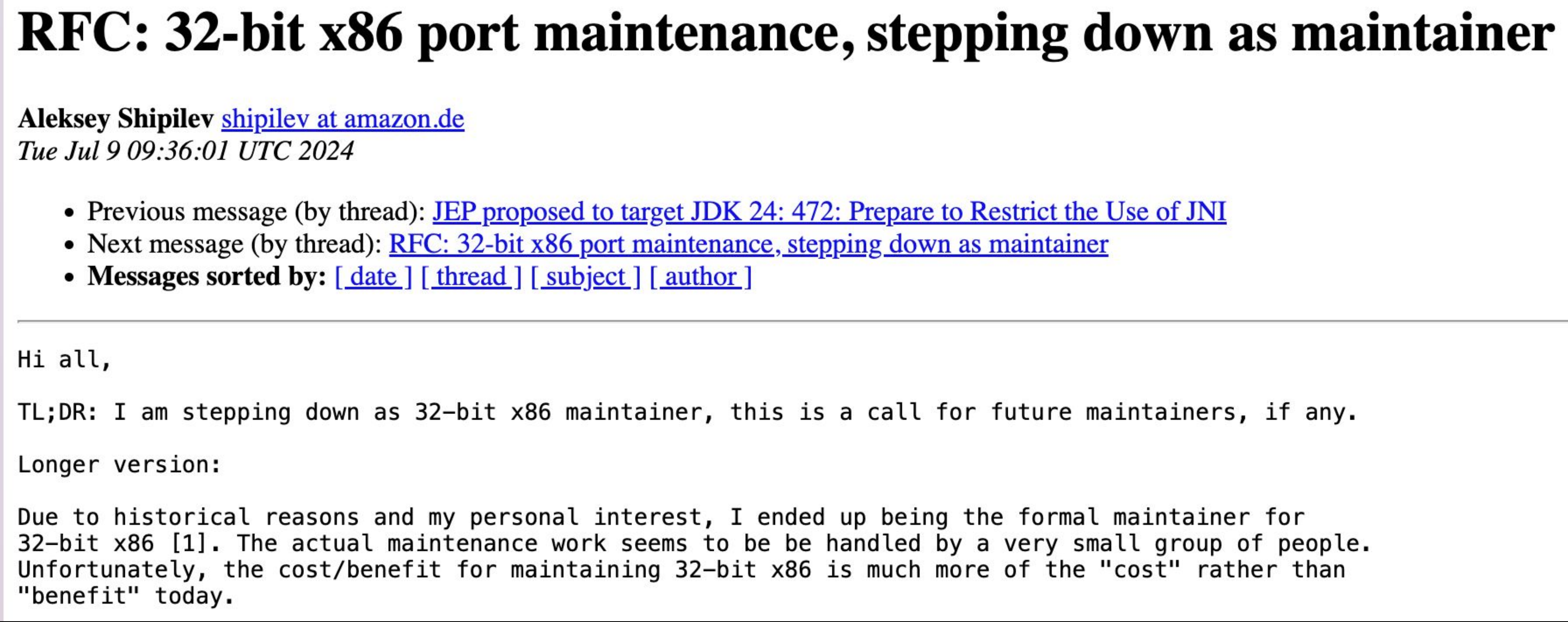
```
Hi all,

TL;DR: I am stepping down as 32-bit x86 maintainer, this is a call for future maintainers, if any.

Longer version:

Due to historical reasons and my personal interest, I ended up being the formal maintainer for
32-bit x86 [1]. The actual maintenance work seems to be be handled by a very small group of people.
Unfortunately, the cost/benefit for maintaining 32-bit x86 is much more of the "cost" rather than
"benefit" today.
```

JEP 501 + JEP 479



RFC: 32-bit x86 port maintenance, stepping down as maintainer

Aleksey Shipilev shipilev at amazon.de
Tue Jul 9 09:36:01 UTC 2024

- Previous message (by thread): JEP proposed to target JDK 24: 472: Prepare to Restrict the Use of JNI
- Next message (by thread): RFC: 32-bit x86 port maintenance, stepping down as maintainer
- **Messages sorted by:** [date] [thread] [subject] [author]

```
Hi all,

TL;DR: I am stepping down as 32-bit x86 maintainer, this is a call for future maintainers, if any.

Longer version:

Due to historical reasons and my personal interest, I ended up being the formal maintainer for
32-bit x86 [1]. The actual maintenance work seems to be be handled by a very small group of people.
Unfortunately, the cost/benefit for maintaining 32-bit x86 is much more of the "cost" rather than
"benefit" today.
```

***There is no pressing industry need for 32-bit x86 with modern JDKs* — We assume that the x86 world has moved firmly to the 64-bit realm.** No new 32-bit-only x86 hardware is being manufactured. The remaining 32-bit x86 deployments are legacies. Industry support has dwindled to match this reality.

490: ZGC: Remove the Non-Generational Mode

Experimental

404: Generational Shenandoah (Experimental)

450: Compact Object Headers (Experimental)

489: Vector API (Ninth Incubator)

489: Vector API (Ninth Incubator)

Зачем нам-бедолагам Vector API?

- SIMD-инструкции в процессорах обрабатывают по 16-32 байт
- SIMD ускоряет парсинг JSON: https://simdjson.org/
- SIMD ускоряет работу UTF, Base64: simdutf/simdutf
- SIMD ускоряет хэш-таблицы: О новых алгоритмах хеш-таблиц

Короче

Это был самый многожепный релиз!

Готовимся
к юбилею Java в мае
релизу Java 25 в сентябре
и Joker в октябре

Мнения, вопросы

Андрей Кулешов
Yandex Infrastructure

Вадим Цесько
01.tech

Владимир Ситников