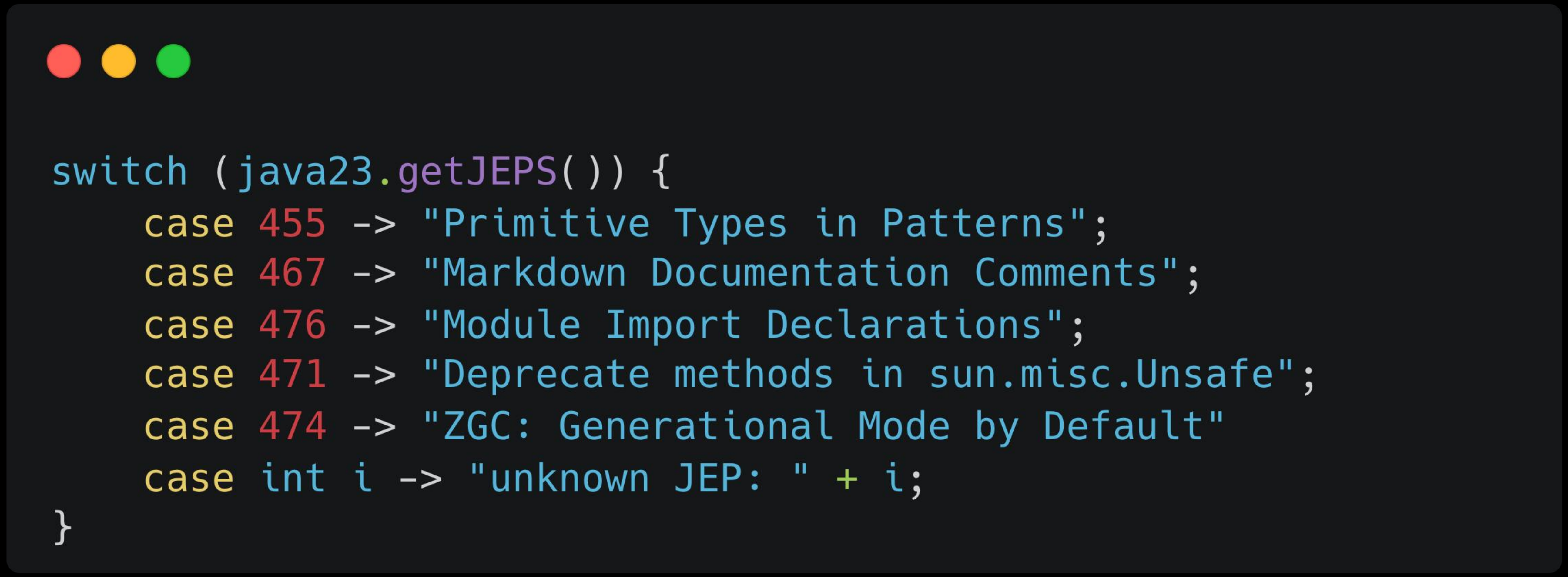
Java 23 – Горячие 🔥 JEP-ы



```
switch (java23.getJEPS()) {
    case 455 -> "Primitive Types in Patterns";
    case 467 -> "Markdown Documentation Comments";
    case 476 -> "Module Import Declarations";
    case 471 -> "Deprecate methods in sun.misc.Unsafe";
    case 474 -> "ZGC: Generational Mode by Default"
    case int i -> "unknown JEP: " + i;
}
```

Андрей Кулешов

Positive Technologies

Андрей Когунь

КРОК

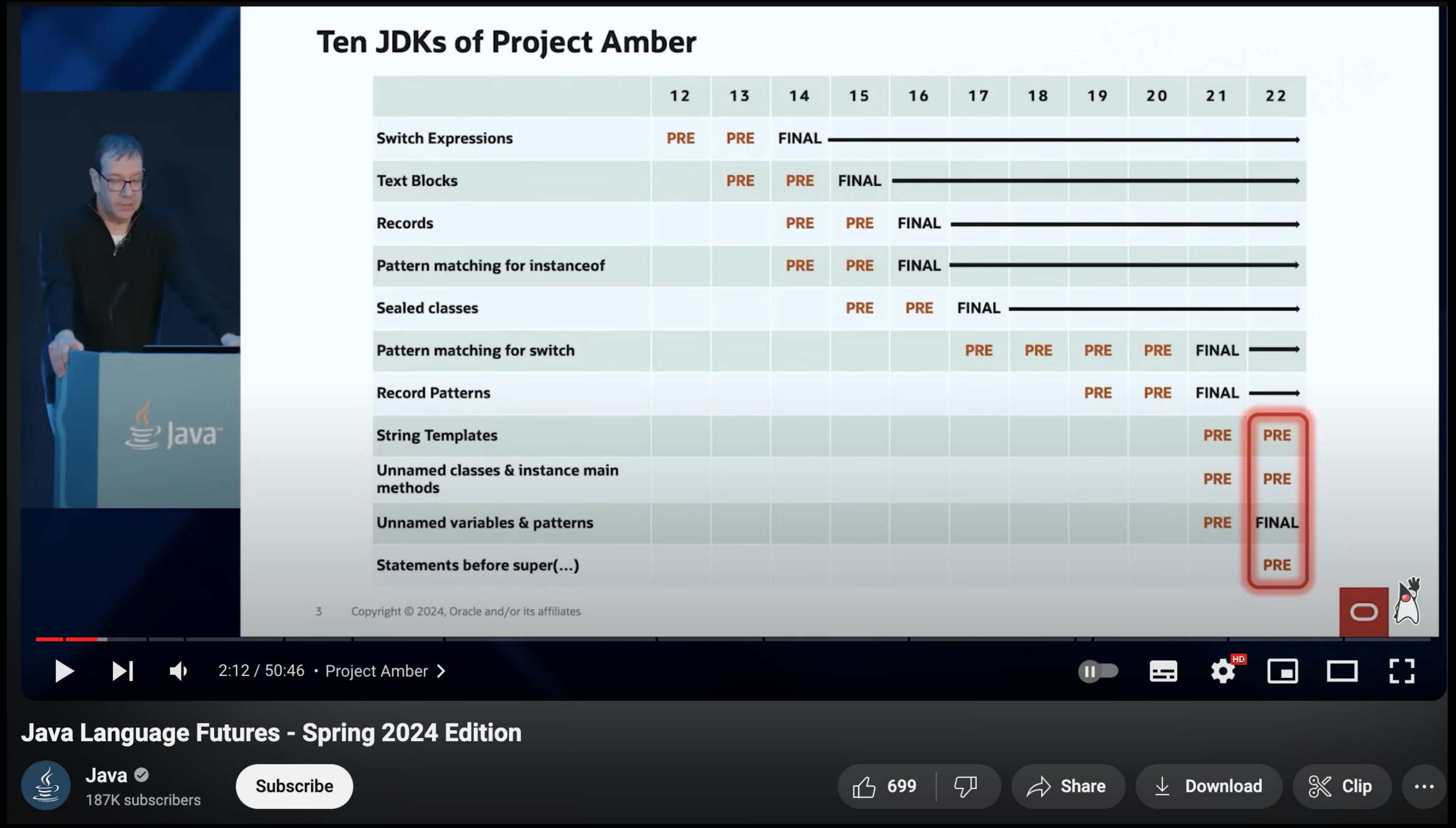
Вадим Цесько

VK

Дмитрий Волыхин

Javaswag

в предыдущих сериях...



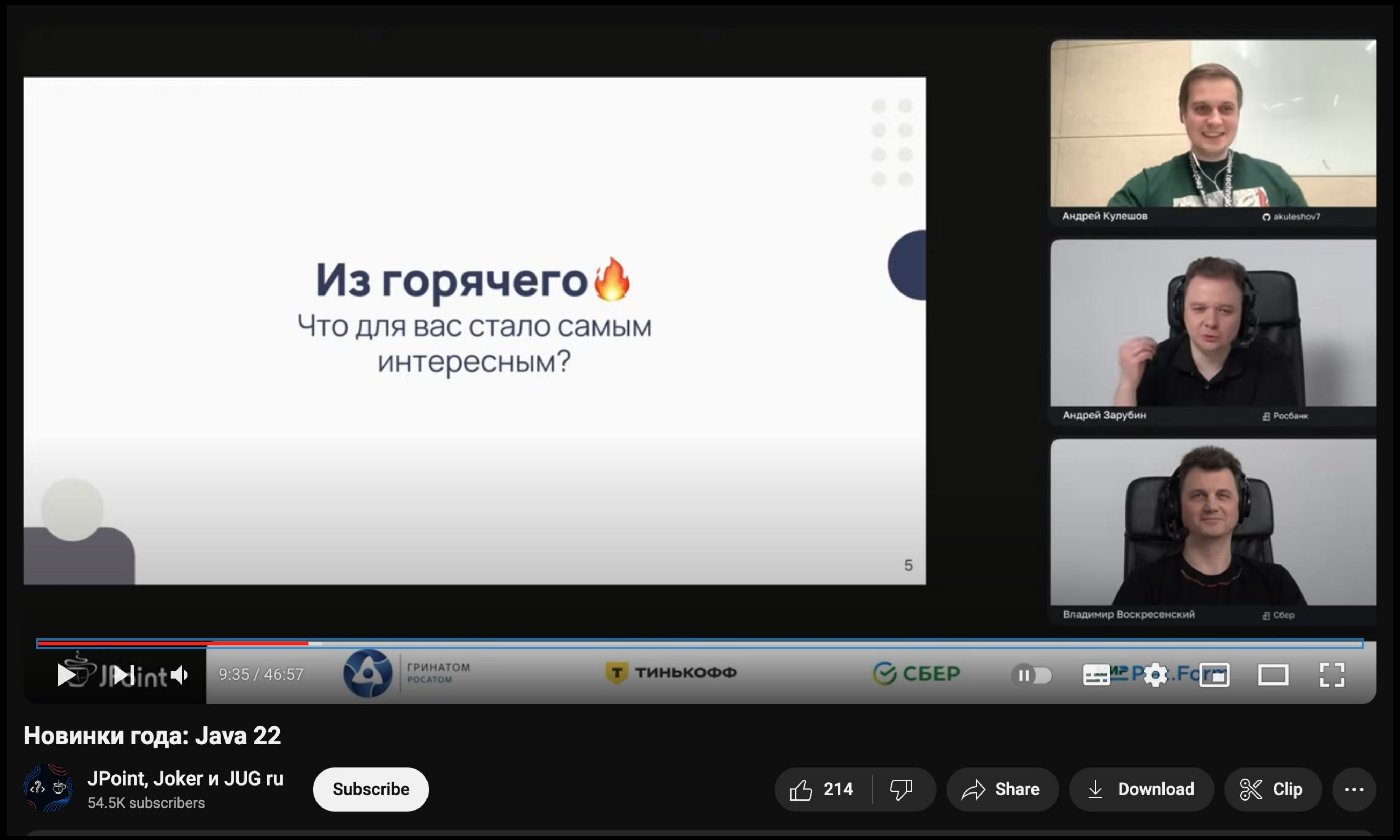
Ten JDKs of Project Amber

	12	13	14	15	16	17	18	19	20	21	22
Switch Expressions	PRE	PRE	FINAL	→							
Text Blocks		PRE	PRE	FINAL	→						
Records			PRE	PRE	FINAL	→					
Pattern matching for instanceof			PRE	PRE	FINAL	→					
Sealed classes				PRE	PRE	FINAL	→				
Pattern matching for switch						PRE	PRE	PRE	PRE	FINAL	→
Record Patterns								PRE	PRE	FINAL	→
String Templates										PRE	PRE
Unnamed classes & instance main methods										PRE	PRE
Unnamed variables & patterns										PRE	FINAL
Statements before super(...)											PRE

3 Copyright © 2024, Oracle and/or its affiliates

Java Language Futures - Spring 2024 Edition

Java
187K subscribers
Subscribe



Из горячего🔥

Что для вас стало самым интересным?

5

Новинки года: Java 22

JPoint, Joker и JUG ru
54.5K subscribers
Subscribe

Java 23 не LTS

467	Markdown Documentation Comments	Релиз
471	Deprecate the Memory-Access Methods in sun.misc.Unsafe for Removal	Релиз
474	ZGC: Generational Mode by Default	Релиз
455	Primitive Types in Patterns, instanceof, and switch	Превью
476	Module Import Declarations	Превью
466	Class-File API	Второе Превью
473	Stream Gatherers	Второе Превью
482	Flexible Constructor Bodies	Второе Превью
477	Implicitely Declared Classes and Instance Main Methods	Третье Превью
481	Scoped Values	Третье Превью
480	Structured Concurrency	Третье Превью
469	Vector API	Восьмое Превью

Java 23 не LTS

467	1	Markdown Documentation Comments	Релиз
471	2	Deprecate the Memory-Access Methods in sun.misc.Unsafe for Removal	Релиз
474	3	ZGC: Generational Mode by Default	Релиз
455	4	Primitive Types in Patterns, instanceof, and switch	Превью
476	5	Module Import Declarations	Превью
466		Class-File API	Второе Превью
473		Stream Gatherers	Второе Превью
482		Flexible Constructor Bodies	Второе Превью
477		Implicitely Declared Classes and Instance Main Methods	Третье Превью
481		Scoped Values	Третье Превью
480		Structured Concurrency	Третье Превью
469		Vector API	Восьмое Превью

1

JEP 467

Markdown Documentation Comments

Было	Стало
JavaDoc	Markdown
текст, <p>	<p> теперь не нужен
примеры кода	примеры кода стали проще
списки	списки лучше
таблицы	таблицы лаконичней
остальной HTML	остальной HTML - понятней
/**	///

JEP 467: Markdown Documentation Comments

Списки - Было

```
/**
 * <ul>
 *     <li>the {@code java.base} module</li>
 *     <li>the {@code java.util} package</li>
 *     <li>a class {@code String}</li>
 *     <li>a class {@code String#CASE_INSENSITIVE_ORDER}</li>
 *     <li>a class {@code String#chars()}</li>
 * </ul>
 */
public void list();
```

Списки - Стало

```
/// - the `java.base`
/// - the `java.util` package
/// - a class `String`
/// - a class `String#CASE_INSENSITIVE_ORDER`
/// - a class `String#chars()`
public void list();
```

Таблицы - Было

Latin	Greek
a	alpha
b	betta
c	gamma

```
/**
 * <table>
 *   <tr>
 *       <th>Latin</th><th>Greek</th>
 *   </tr>
 *   <tr>
 *     <td>a</td><td>alpha</td>
 *   </tr>
 *   <tr>
 *     <td>b</td><td>betta</td>
 *   </tr>
 *    <tr>
 *     <td>c</td><td>gamma</td>
 *   </tr>
 * </table>
 */
public void printTable();
```

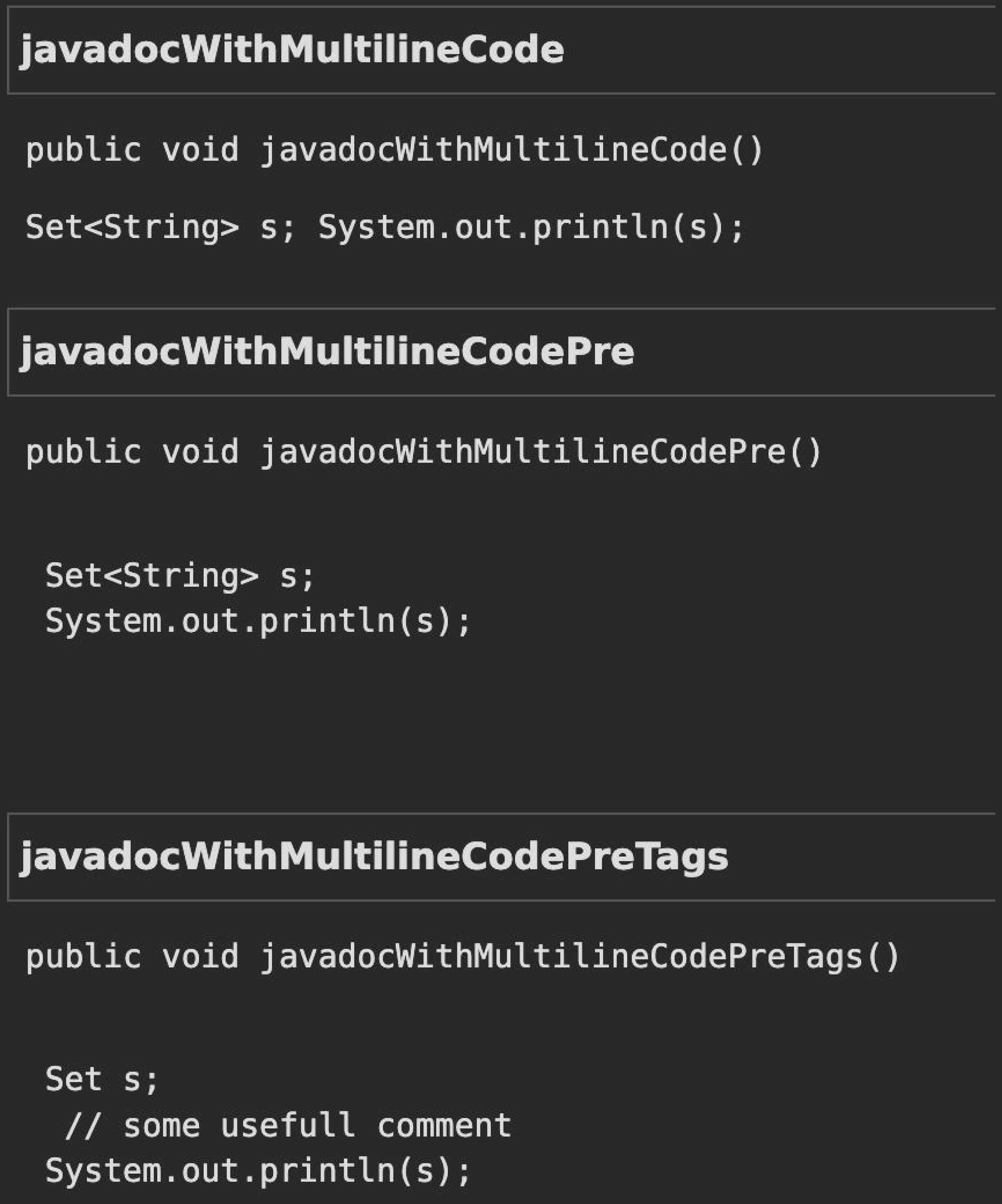
Таблицы - Стало

Markdown

```
/// | Latin | Greek |
/// |-------|-------|
/// | a     | alpha |
/// | b     | beta  |
/// | c     | gamma |
public void printTable();
```

Код - Было

```
/**
* {@code
* Set<String> s;
* System.out.println(s);
* }
*/
public void javadocWithMultilineCode();


/**
* <pre>
* {@code
* Set<String> s;
* System.out.println(s);
* }
* </pre>
*/
public void javadocWithMultilineCodePre();
```



javadocWithMultilineCode

```
public void javadocWithMultilineCode()

Set<String> s; System.out.println(s);
```

javadocWithMultilineCodePre

```
public void javadocWithMultilineCodePre()


 Set<String> s;
 System.out.println(s);
```

javadocWithMultilineCodePreTags

```
public void javadocWithMultilineCodePreTags()


 Set s;
  // some usefull comment
 System.out.println(s);
```

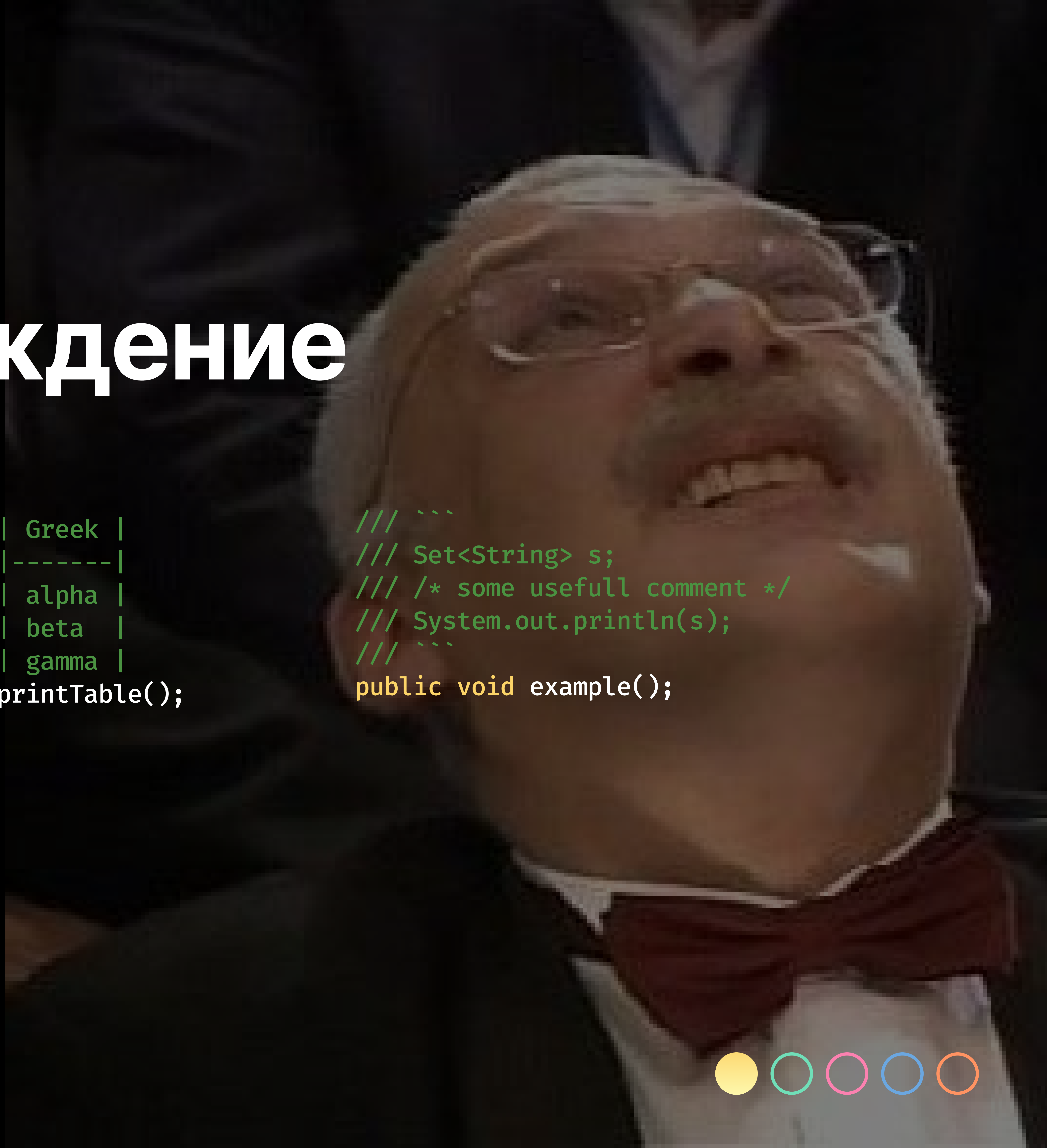
Код - Было

```
/**
* <pre>
* <code>
* Set<String> s;
*  /* some usefull comment */
* System.out.println(s);
* </code>
* </pre>
*/
public void javadocWithComment();
```

Код - Стало

```
/// ```
/// Set<String> s;
/// /* some usefull comment */
/// System.out.println(s);
/// ```
public void example();
```

Обсуждение



```
/// - the `java.base`
/// - the `java.util` package
/// - a class `String`
public void list();
```

```
/// | Latin | Greek |
/// |-------|-------|
/// | a     | alpha |
/// | b     | beta  |
/// | c     | gamma |
public void printTable();
```

```
/// ```
/// Set<String> s;
/// /* some usefull comment */
/// System.out.println(s);
/// ```
public void example();
```

2

JEP 471

Deprecate the Memory-Access Methods in sun.misc.Unsafe for Removal

Java 9

JEP 193: Variable Handles

On-heap methods

```
long objectFieldOffset(Field f)
long staticFieldOffset(Field f)
Object staticFieldBase(Field f)
int arrayBaseOffset(Class<?> arrayClass)
int arrayIndexScale(Class<?> arrayClass)
```

Java 22

JEP 454: Foreign Function & Memory API

Off-heap methods

```
long allocateMemory(long bytes);
long reallocateMemory(long address, long bytes);
void freeMemory(long address);
void invokeCleaner(java.nio.ByteBuffer directBuffer);
void setMemory(long address, long bytes, byte value);
void copyMemory(long srcAddress, long destAddress, long bytes);
get[Type](long address);
void put[Type](long address, [type] x);
long getAddress(long address);
void putAddress(long address, long x);
int addressSize();
```

JEP 471: Deprecate the Memory-Access Methods in sun.misc.Unsafe for Removal

Новый ключ --sun-misc-unsafe-memory-access ={allow|warn|debug|deny}

Фаза 1
Deprecate for removal

Фаза 2
Runtime warning

Фаза 3 Java 26 выйдет в марте 2026
Throw Exception by default

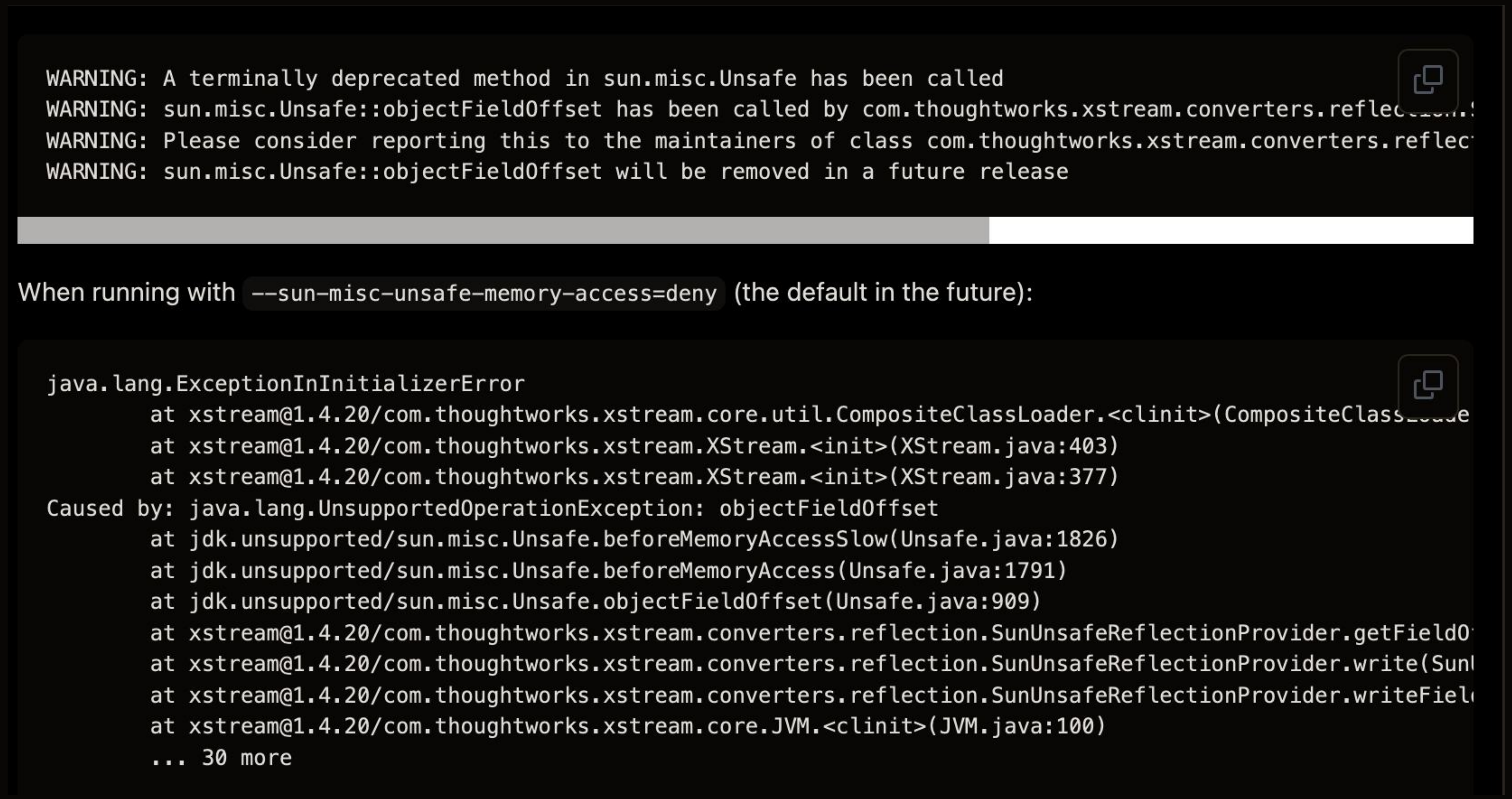
Фаза 4 и 5
Remove methods

JEP 471: Deprecate the Memory-Access Methods in sun.misc.Unsafe for Removal

Возможно, вы не знаете что у вас есть Unsafe

https://github.com/x-stream/xstream/issues/362

https://github.com/eclipse-equinox/equinox/issues/489



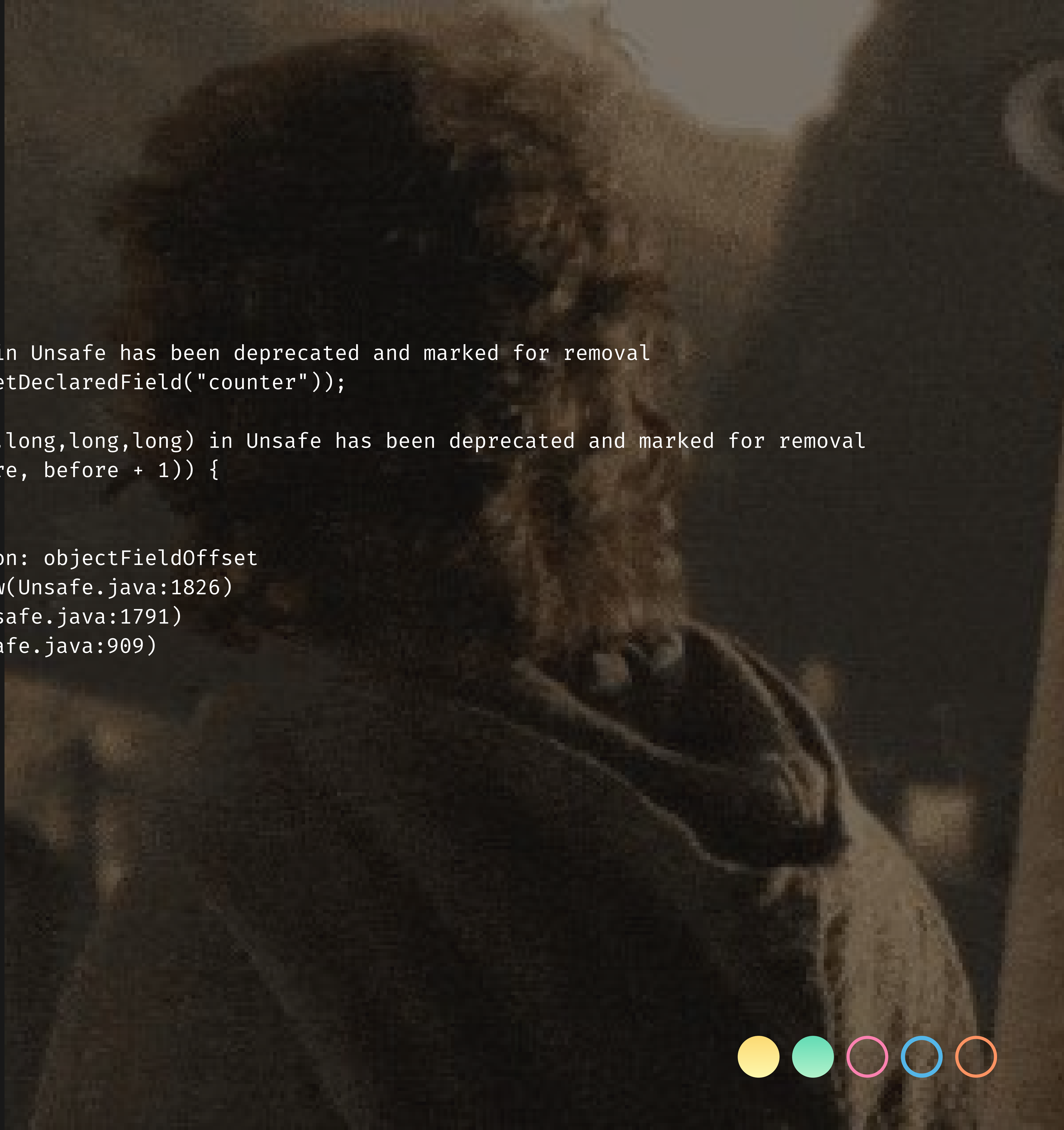
```
WARNING: A terminally deprecated method in sun.misc.Unsafe has been called
WARNING: sun.misc.Unsafe::objectFieldOffset has been called by com.thoughtworks.xstream.converters.reflection.S
WARNING: Please consider reporting this to the maintainers of class com.thoughtworks.xstream.converters.reflec
WARNING: sun.misc.Unsafe::objectFieldOffset will be removed in a future release
```

When running with `--sun-misc-unsafe-memory-access=deny` (the default in the future):

```
java.lang.ExceptionInInitializerError
        at xstream@1.4.20/com.thoughtworks.xstream.core.util.CompositeClassLoader.<clinit>(CompositeClassLoade
        at xstream@1.4.20/com.thoughtworks.xstream.XStream.<init>(XStream.java:403)
        at xstream@1.4.20/com.thoughtworks.xstream.XStream.<init>(XStream.java:377)
Caused by: java.lang.UnsupportedOperationException: objectFieldOffset
        at jdk.unsupported/sun.misc.Unsafe.beforeMemoryAccessSlow(Unsafe.java:1826)
        at jdk.unsupported/sun.misc.Unsafe.beforeMemoryAccess(Unsafe.java:1791)
        at jdk.unsupported/sun.misc.Unsafe.objectFieldOffset(Unsafe.java:909)
        at xstream@1.4.20/com.thoughtworks.xstream.converters.reflection.SunUnsafeReflectionProvider.getFieldO
        at xstream@1.4.20/com.thoughtworks.xstream.converters.reflection.SunUnsafeReflectionProvider.write(Sun
        at xstream@1.4.20/com.thoughtworks.xstream.converters.reflection.SunUnsafeReflectionProvider.writeFiel
        at xstream@1.4.20/com.thoughtworks.xstream.core.JVM.<clinit>(JVM.java:100)
        ... 30 more
```

JEP 471: Deprecate the Memory-Access Methods in sun.misc.Unsafe for Removal

Обсуждение



```
 java --sun-misc-unsafe-memory-access=deny UnsafeMain.java
UnsafeMain.java:31: warning: [removal] objectFieldOffset(Field) in Unsafe has been deprecated and marked for removal
            offset = unsafe.objectFieldOffset(CASCounter.class.getDeclaredField("counter"));
                           ^
UnsafeMain.java:36: warning: [removal] compareAndSwapLong(Object,long,long,long) in Unsafe has been deprecated and marked for removal
            while (!unsafe.compareAndSwapLong(this, offset, before, before + 1)) {
                          ^
2 warnings
Exception in thread "main" java.lang.UnsupportedOperationException: objectFieldOffset
        at jdk.unsupported/sun.misc.Unsafe.beforeMemoryAccessSlow(Unsafe.java:1826)
        at jdk.unsupported/sun.misc.Unsafe.beforeMemoryAccess(Unsafe.java:1791)
        at jdk.unsupported/sun.misc.Unsafe.objectFieldOffset(Unsafe.java:909)
        at UnsafeMain$CASCounter.<init>(UnsafeMain.java:31)
        at UnsafeMain.main(UnsafeMain.java:9)
```

JEP 471: Deprecate the Memory-Access Methods in sun.misc.Unsafe for Removal

3

JEP 474

ZGC: Generational Mode by Default

Description

Make Generational ZGC the default mode of ZGC by changing the default value of the ZGenerational option from **false** to **true**. Deprecate the non-generational mode by deprecating the ZGenerational option.
After these changes the following behavior will be observed based on the provided command-line arguments:

- **-XX:+UseZGC**
  - Generational ZGC is used.

- **-XX:+UseZGC -XX:+ZGenerational**
  - Generational ZGC is used.
  - A warning that the ZGenerational option is deprecated is issued.

- **-XX:+UseZGC -XX:-ZGenerational**
  - Non-generational ZGC is used.
  - A warning that the ZGenerational option is deprecated is issued.
  - A warning that the non-generational mode is deprecated for removal is issued.

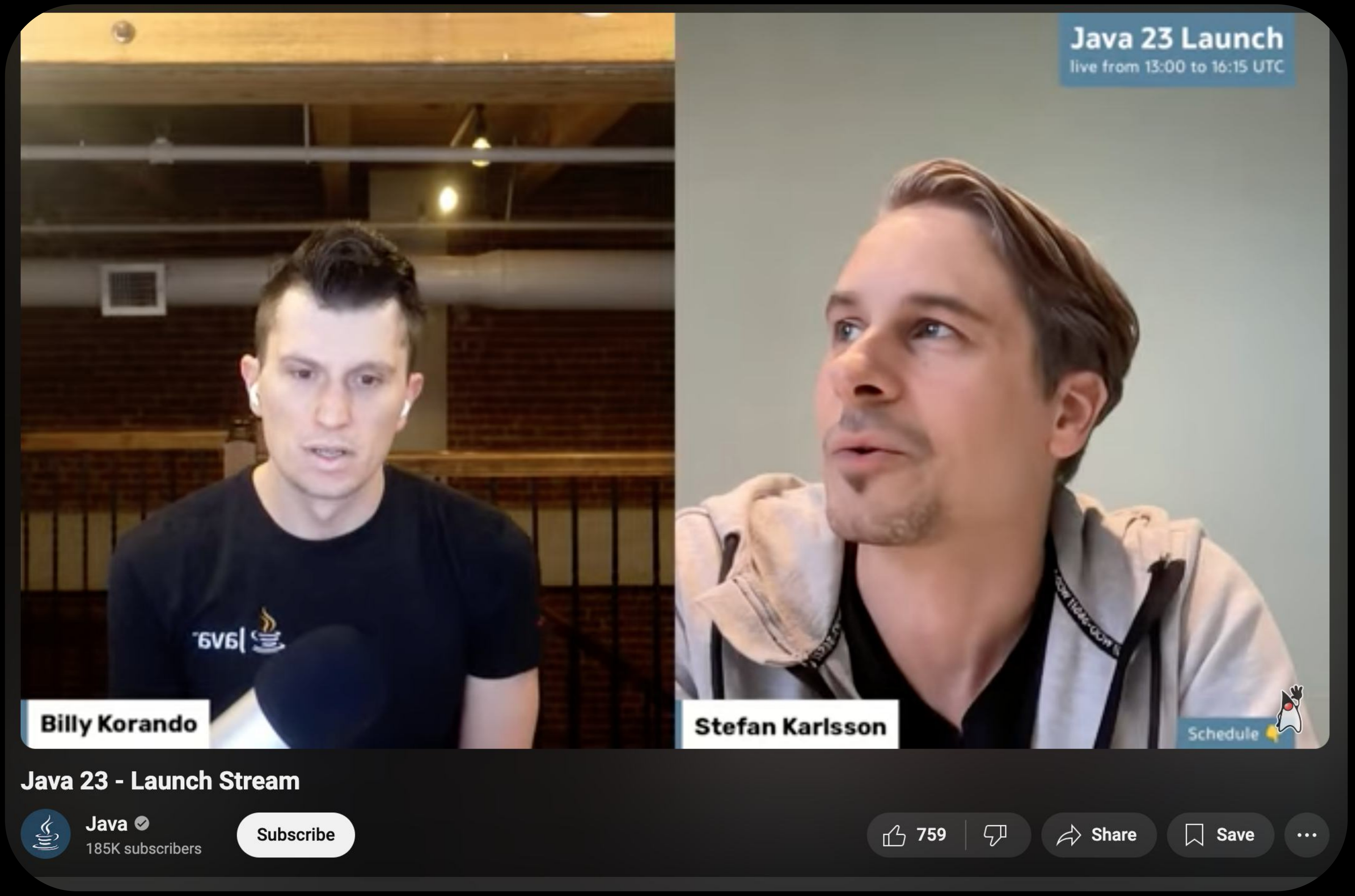
Summary

Switch the default mode of the Z Garbage Collector (ZGC) to the generational mode. Deprecate the non-generational mode, with the intent to remove it in a future release.

Motivation

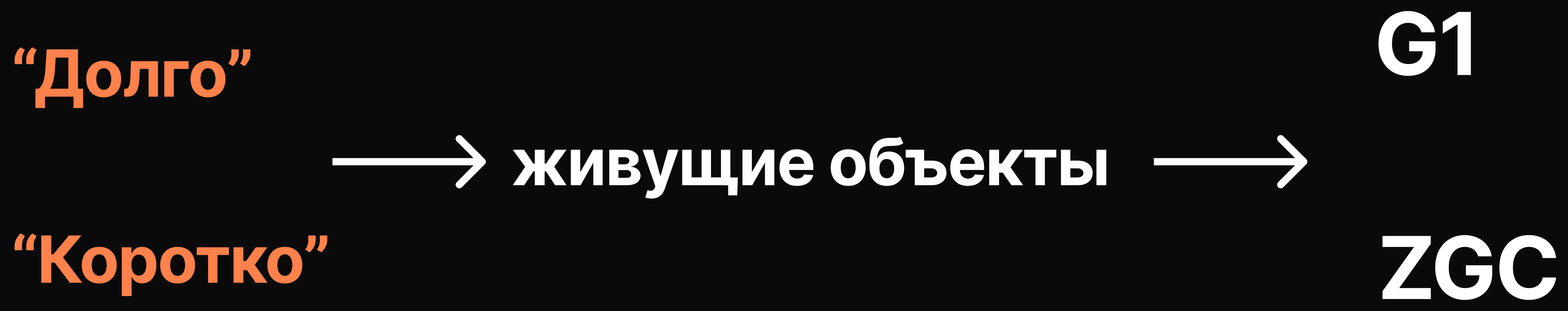
Maintaining non-generational ZGC slows the development of new features. As stated in JEP 439: Generational ZGC:

> *Generational ZGC should be a better solution for most use cases than non-generational ZGC. We should eventually be able to replace the latter with the former in order to reduce long-term maintenance costs.*



- Каждый должен попробовать ZGC
- Короткие транзакции, например вызовы микросервисов
- Микросекунды, в отличие от других GC с лэтенси 10 миллисекунд
- Убрать все параметры GC и выстаавить Xmx – это всё
- Смотрите на метрики своего приложения

Совет от Нетфликса

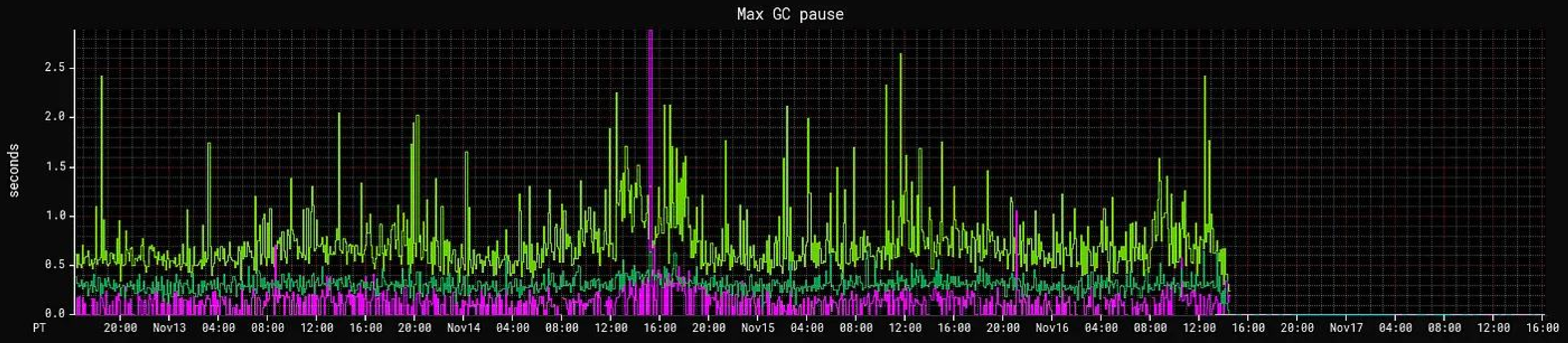

“Долго”

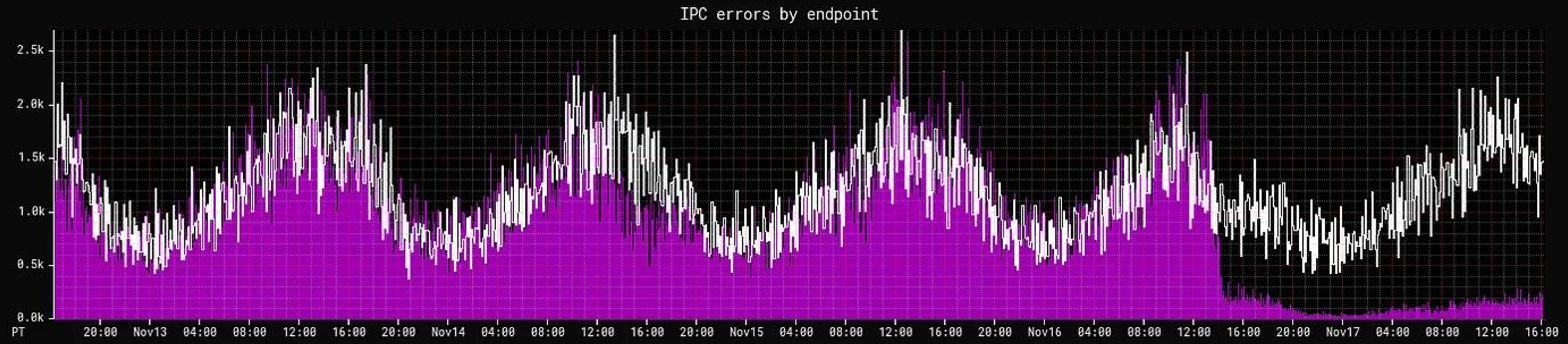
“Коротко”

живущие объекты

G1

ZGC


Max GC pause


IPC errors by endpoint

Обсуждение

Make Generational ZGC the default mode of ZGC by changing the default value of the ZGenerational option from false to true

JEP 474: ZGC: Generational Mode by Default

4

JEP 455

Primitive Types in Patterns, instanceof, and switch

Превью

```java
int status = x.status();
switch (status) {
    case 0 -> “okay”;
    case 1 -> “warning”;
    case 2 -> “error”;
    default -> “unknown status: ” + status;
}
```

JEP 455: Primitive Types in Patterns, instanceof, and switch (Preview)

Что значит double instanceof long?

```
private static boolean isInteger(double d) {
    return d instanceof long;
}
```

JEP 455: Primitive Types in Patterns, instanceof, and switch (Preview)

Что значит double instanceof long?

```
private static boolean isInteger(double d) {
    return d instanceof long;
    // return d == (long d);
}
```

JEP 455: Primitive Types in Patterns, instanceof, and switch (Preview)

Java 23 2024

JEP 455: Primitive Types in Patterns, instanceof, and switch (Preview)

Goals

- Enable uniform data exploration by allowing type patterns for all types, whether primitive or reference.
- Align type patterns with instanceof, and align instanceof with safe casting.
- Allow pattern matching to use primitive type patterns in both nested and top-level contexts.
- Provide easy-to-use constructs that eliminate the risk of losing information due to unsafe casts.
- Following the enhancements to switch in **Java 5** (enum switch) and **Java 7** (string switch), allow switch to process values of any primitive type.

Паттерн Матчинг

Java 14 2020

JEP 361: Switch expressions

```
switch (shape) {
  case Point p -> processY(p.y());
  // other branches
}
```

Java 16-17 2021

JEP 394: Pattern Matching for instanceof

```
if (shape instanceof Point p) {
  processY(p.y());
}
```

JEP 406: Pattern Matching for switch (preview)

```
switch (shape) {
  case Point p when p == 0 -> processY(p.y());
  case Point p -> process(p.x(), p.y());
  // other branches
}
```

Java 21 2023

JEP 440: Record Patterns

JEP 443: Unnamed Pattern and Variables (Preview)

```
switch (shape) {
    case Point(int _, int y) -> processY(y);
}

record Point(int x, int y) {}
```

JEP 455: Primitive Types in Patterns, instanceof, and switch (Preview)

Primitive паттерны

NEW!

```
int status = x.status();
switch (status) {
    case 0 -> “okay”;
    case 1 -> “warning”;
    case 2 -> “error”;
    default -> “unknown status: ” + status;
}
```

JEP 455: Primitive Types in Patterns, instanceof, and switch (Preview)

Пример #1

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

// Since Java 23

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

JEP 455: Primitive Types in Patterns, instanceof, and switch (Preview)

Пример #2

```java
var i = 256;
switch (i) {
    case byte b -> System.out.println("byte: " + b);
    case short s -> System.out.println("short: " + s);
    default -> System.out.println("int: " + i);
}
```

JEP 455: Primitive Types in Patterns, instanceof, and switch (Preview)

Пример #3

```java
byte b = 42;
b instanceof int;         // true (unconditionally exact)

int i = 42;
i instanceof byte;        // true (exact)

int i = 1000;
i instanceof byte;        // false (not exact)

int i = 16_777_217;       // 2^24 + 1
i instanceof float;       // false (not exact)
i instanceof double;      // true (unconditionally exact)
i instanceof Integer;     // true (unconditionally exact)
i instanceof Number;      // true (unconditionally exact)

float f = 1000.0f;
f instanceof byte;        // false
f instanceof int;         // true (exact)
f instanceof double;      // true (unconditionally exact)
```

```java
double d = 1000.0d;
d instanceof byte;        // false
d instanceof int;         // true (exact)
d instanceof float;       // true (exact)

Integer ii = 1000;
ii instanceof int;        // true (exact)
ii instanceof float;      // true (exact)
ii instanceof double;     // true (exact)

Integer ii = 16_777_217;
ii instanceof float;      // false (not exact)
ii instanceof double;     // true (exact)
```

JEP 455: Primitive Types in Patterns, instanceof, and switch (Preview)

Пример #3

```
byte b = 42;
b instanceof int;         // true (unconditionally exact)

int i = 42;
i instanceof byte;        // true (exact)

int i = 1000;
i instanceof byte;        // false (not exact)

int i = 16_777_217;       // 2^24 + 1
i instanceof float;       // false (not exact)
i instanceof double;      // true (unconditionally exact)
i instanceof Integer;     // true (unconditionally exact)
i instanceof Number;      // true (unconditionally exact)

float f = 1000.0f;
f instanceof byte;        // false
f instanceof int;         // true (exact)
f instanceof double;      // true (unconditionally exact)
```

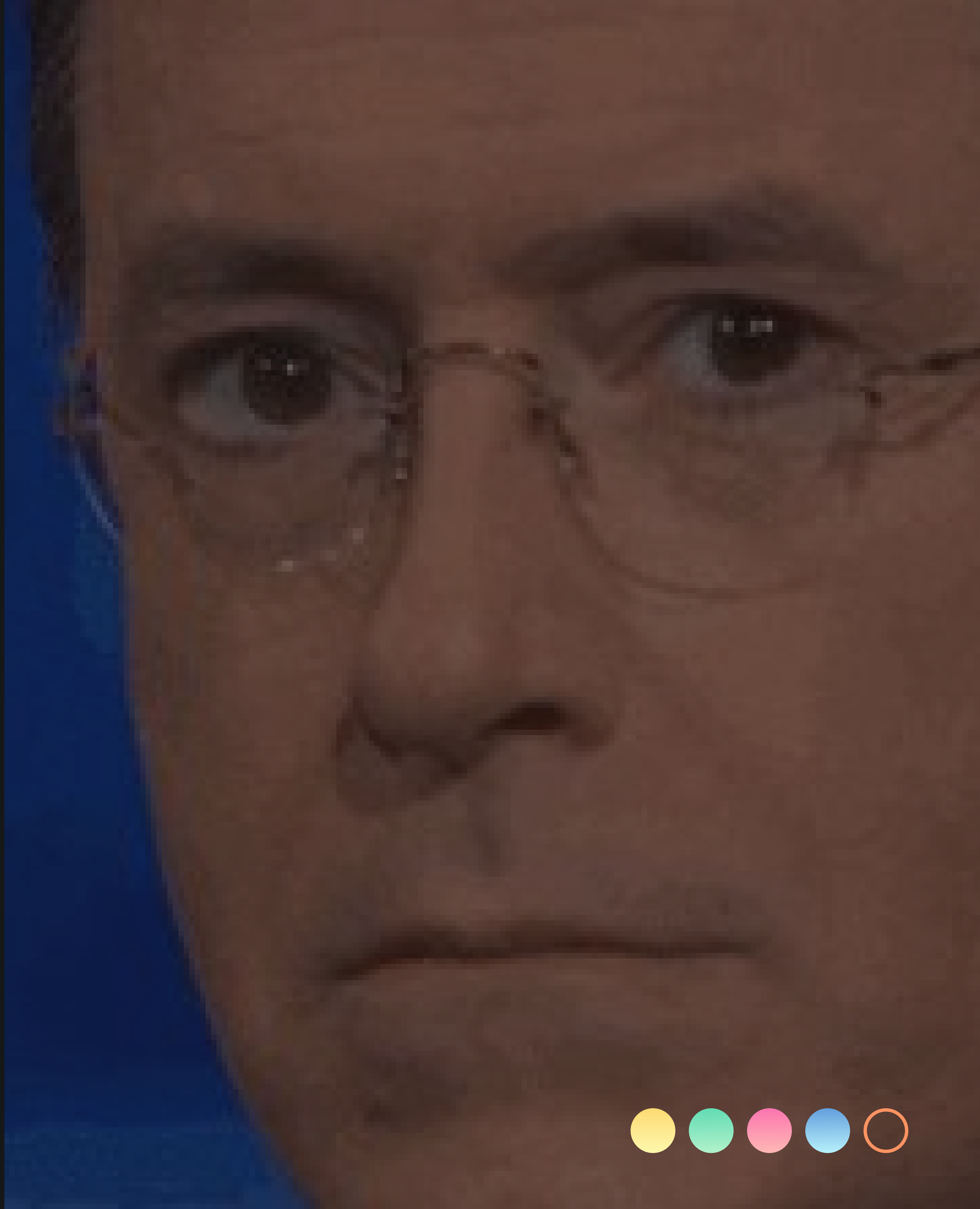
```
double d = 1000.0d;
d instanceof byte;          // false
d instanceof int;           // true (exact)
d instanceof float;         // true (exact)

Integer ii = 1000;
ii instanceof int;          // true (exact)
ii instanceof float;        // true (exact)
ii instanceof double;       // true (exact)

Integer ii = 16_777_217;
ii instanceof float;        // false (not exact)
ii instanceof double;       // true (exact)
```

JEP 455: Primitive Types in Patterns, instanceof, and switch (Preview)

Обсуждение

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

```
var i = 256;
switch (i) {
    case byte b -> System.out.println("byte: " + b);
    case short s -> System.out.println("short: " + s);
    default -> System.out.println("int: " + i);
}
```



JEP 455: Primitive Types in Patterns, instanceof, and switch (Preview)

5

JEP 476

Module Import Declarations (Preview)

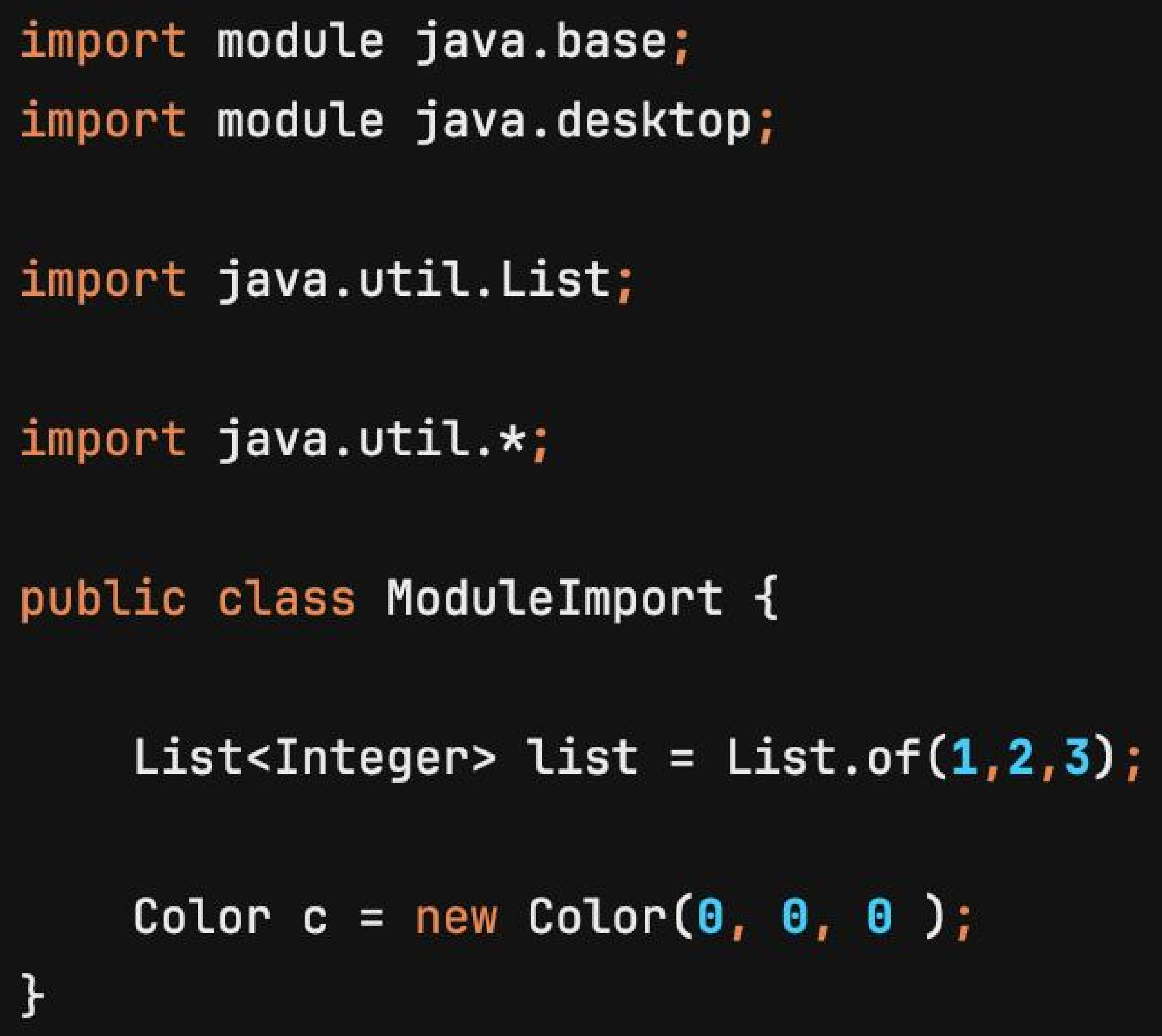
https://openjdk.org/jeps/476

```java
import java.util.*;
```

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

JEP 476: Module Import Declarations (Preview)

Конфликты при импорте



```java
import module java.base;
import module java.desktop;

import java.util.List;

import java.util.*;

public class ModuleImport {

    List<Integer> list = List.of(1,2,3);

    Color c = new Color(0, 0, 0 );
}
```

JEP 476: Module Import Declarations (Preview)

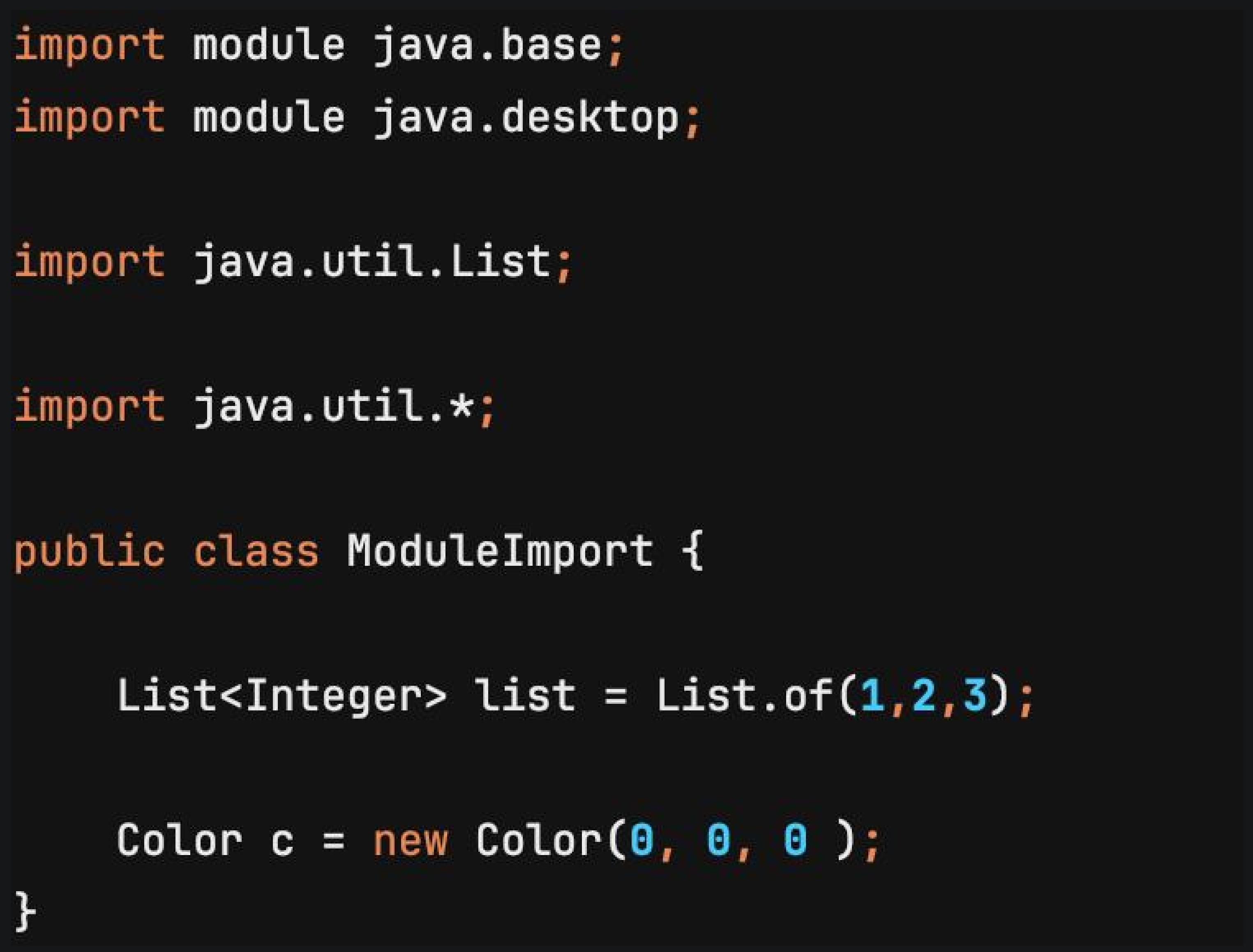
Неявный импорт java.base

```java
// Main.java - нет явных импортов!
void main() {
    List<?> dates = Stream
        .of(1, 2, 23, 29)
        .map(BigDecimal::new)
        .map(day -> LocalDate.of(
            2024,
            RandomGenerator.getDefault()
                .nextInt(11) + 1,
            day.intValue()))
        .toList();

    System.out.println(dates);
}
```

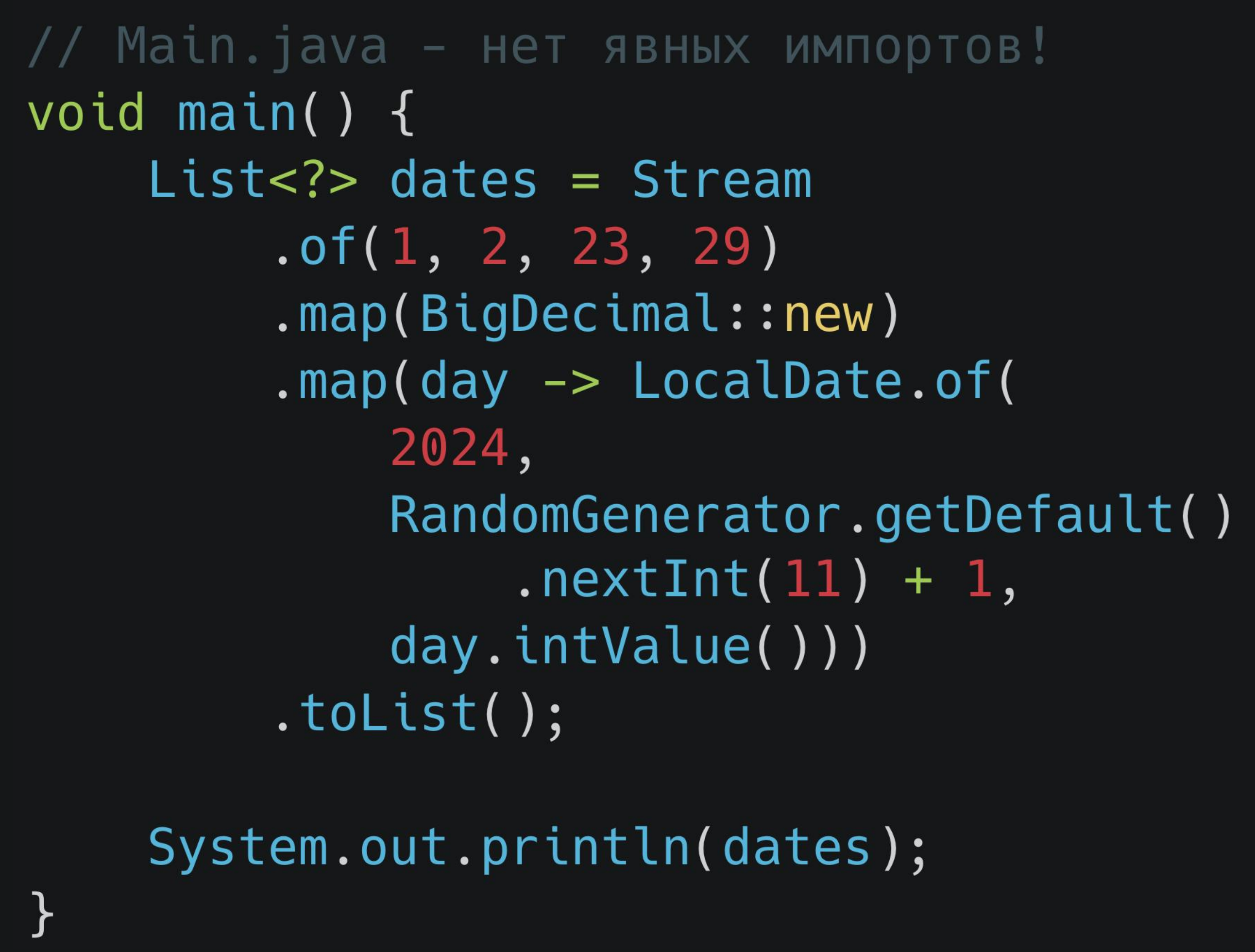
JEP 476: Module Import Declarations (Preview)

JEP 476: Module Import Declarations (Preview)

Обсуждение



```java
import module java.base;
import module java.desktop;

import java.util.List;

import java.util.*;

public class ModuleImport {

    List<Integer> list = List.of(1,2,3);

    Color c = new Color(0, 0, 0 );
}
```



```java
// Main.java - нет явных импортов!
void main() {
    List<?> dates = Stream
        .of(1, 2, 23, 29)
        .map(BigDecimal::new)
        .map(day -> LocalDate.of(
            2024,
            RandomGenerator.getDefault()
                .nextInt(11) + 1,
            day.intValue()))
        .toList();

    System.out.println(dates);
}
```

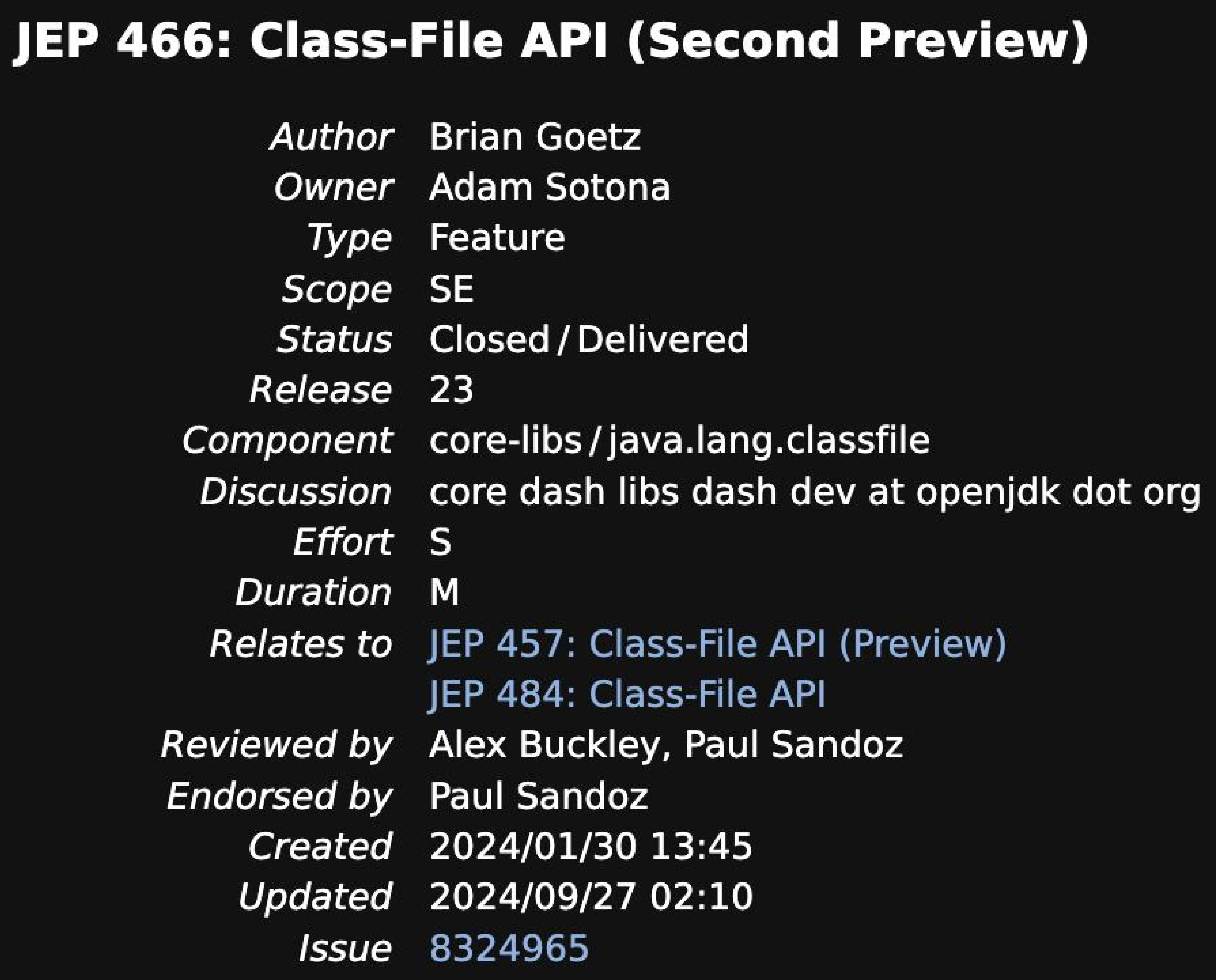
6

JEP 466

Class-File API

Второе превью

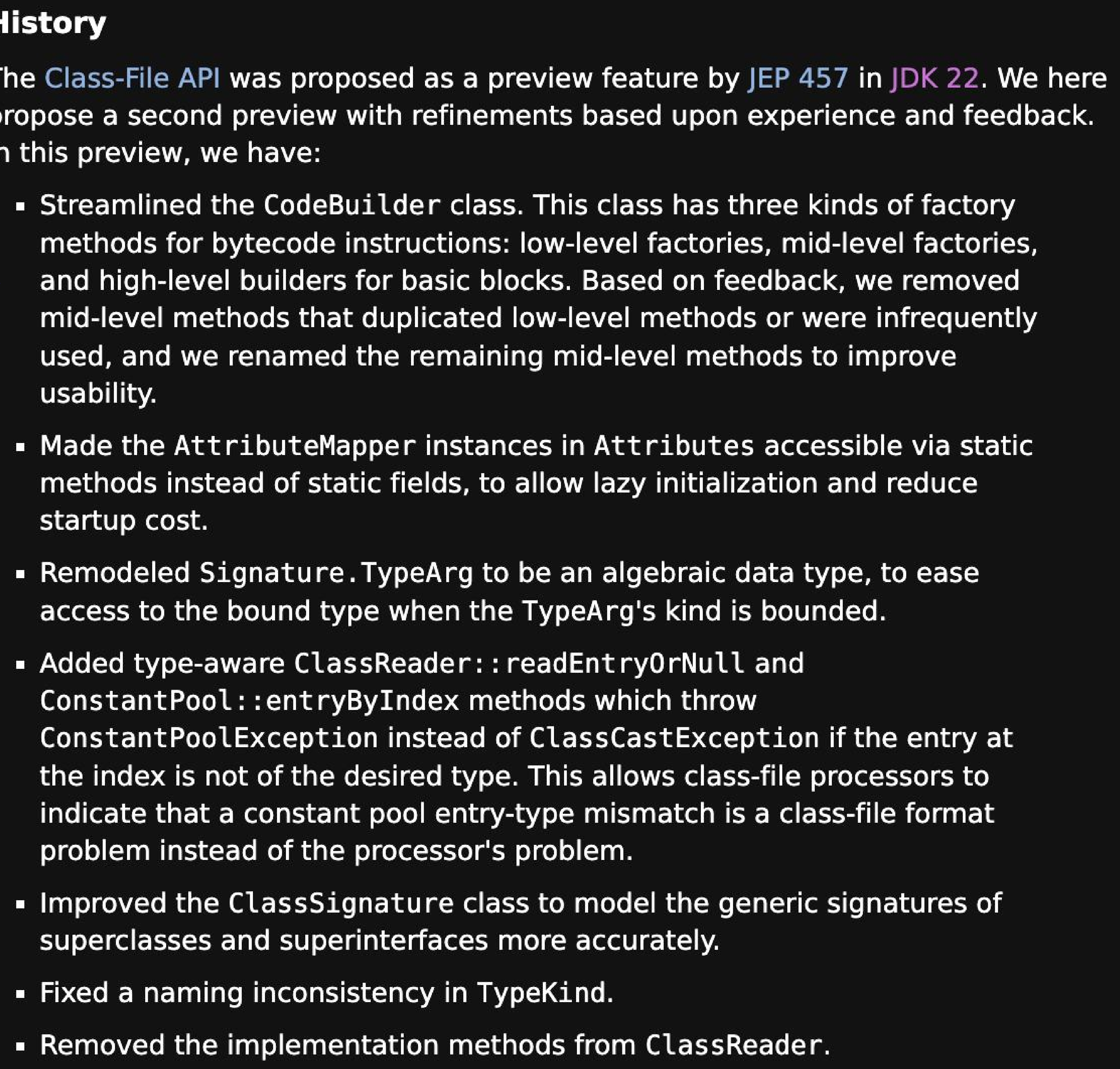
https://openjdk.org/jeps/476



JEP 466: Class-File API (Second Preview)

Author	Brian Goetz
Owner	Adam Sotona
Type	Feature
Scope	SE
Status	Closed / Delivered
Release	23
Component	core-libs / java.lang.classfile
Discussion	core dash libs dash dev at openjdk dot org
Effort	S
Duration	M
Relates to	JEP 457: Class-File API (Preview) JEP 484: Class-File API
Reviewed by	Alex Buckley, Paul Sandoz
Endorsed by	Paul Sandoz
Created	2024/01/30 13:45
Updated	2024/09/27 02:10
Issue	8324965

NEW!



History

The Class-File API was proposed as a preview feature by JEP 457 in JDK 22. We here propose a second preview with refinements based upon experience and feedback. In this preview, we have:

- Streamlined the `CodeBuilder` class. This class has three kinds of factory methods for bytecode instructions: low-level factories, mid-level factories, and high-level builders for basic blocks. Based on feedback, we removed mid-level methods that duplicated low-level methods or were infrequently used, and we renamed the remaining mid-level methods to improve usability.
- Made the `AttributeMapper` instances in `Attributes` accessible via static methods instead of static fields, to allow lazy initialization and reduce startup cost.
- Remodeled `Signature.TypeArg` to be an algebraic data type, to ease access to the bound type when the `TypeArg`'s kind is bounded.
- Added type-aware `ClassReader::readEntryOrNull` and `ConstantPool::entryByIndex` methods which throw `ConstantPoolException` instead of `ClassCastException` if the entry at the index is not of the desired type. This allows class-file processors to indicate that a constant pool entry-type mismatch is a class-file format problem instead of the processor's problem.
- Improved the `ClassSignature` class to model the generic signatures of superclasses and superinterfaces more accurately.
- Fixed a naming inconsistency in `TypeKind`.
- Removed the implementation methods from `ClassReader`.

JEP 466: Class-File API (Second Preview)

```
ClassFile cf = ClassFile.of();
ClassModel classModel = cf.parse(bytes);
byte[] newBytes = cf.build(classModel.thisClass().asSymbol(),
        classBuilder -> {
            for (ClassElement ce : classModel) {
                if (ce instanceof MethodModel mm) {
                    classBuilder.withMethod(mm.methodName(), mm.methodType(),
                            mm.flags().flagsMask(), methodBuilder -> {
                                for (MethodElement me : mm) {
                                    if (me instanceof CodeModel codeModel) {
                                        methodBuilder.withCode(codeBuilder -> {
                                            for (CodeElement e : codeModel) {
                                                switch (e) {
                                                    case InvokeInstruction i
                                                            when i.owner().asInternalName().equals("Foo")) ->
                                                        codeBuilder.invoke(i.opcode(),
                                                                ClassDesc.of("Bar"),
                                                                i.name(), i.type());
                                                        default -> codeBuilder.with(e);
                                                }
                                            }
                                        });
                                    }
                                    else
                                        methodBuilder.with(me);
                                }
                            });
                }
                else
                    classBuilder.with(ce);
            }
        });
```

JEP 466: Class-File API (Second Preview)

7

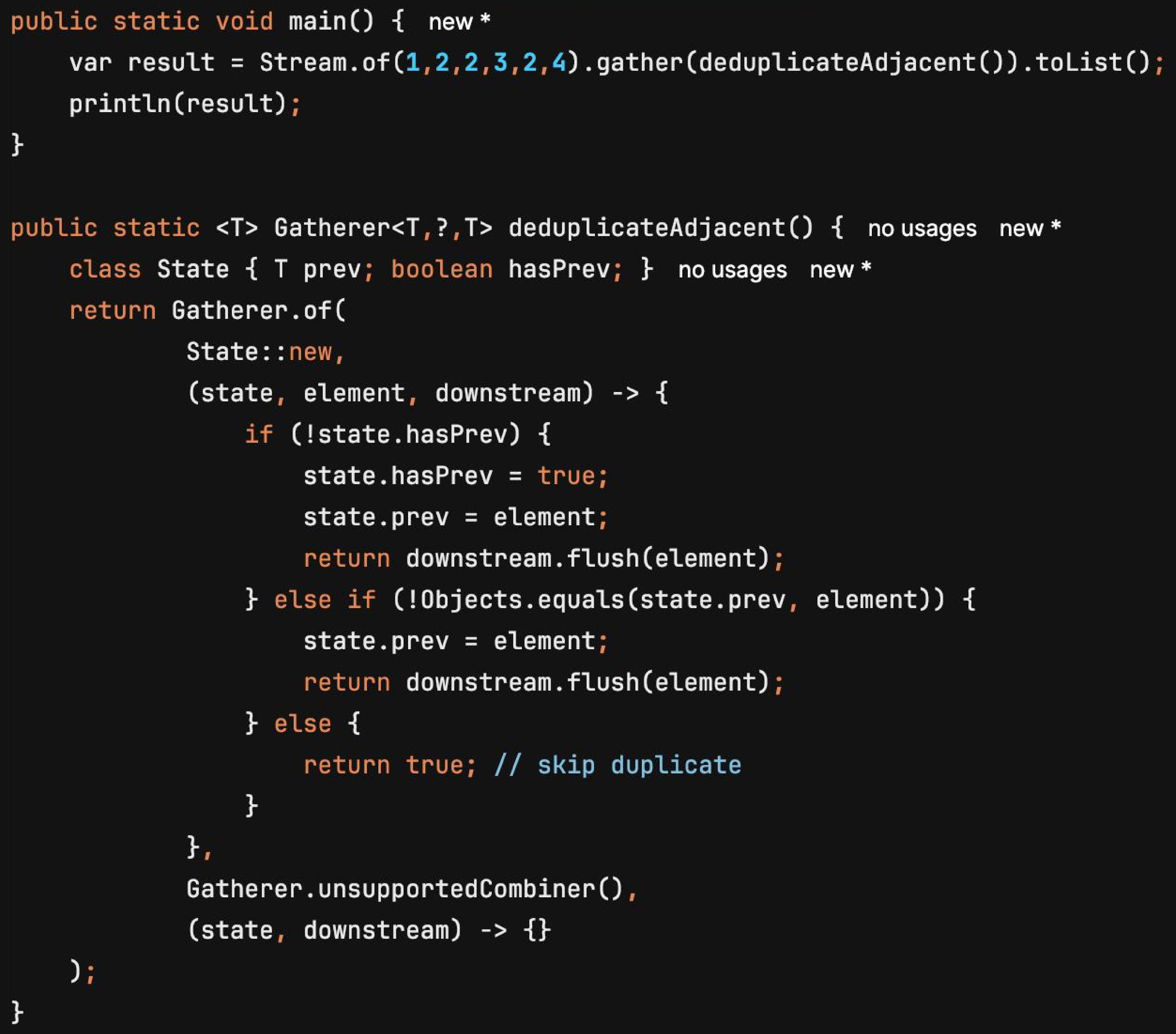
JEP 474

Stream Gatherers

Второе превью без изменений

Gatherer умеет

- обрабатывать поток в режиме
  - one-to-one
  - one-to-many
  - many-to-one
  - many-to-many
- хранить состояние
- работать параллельно



```
public static void main() {  new *
    var result = Stream.of(1,2,2,3,2,4).gather(deduplicateAdjacent()).toList();
    println(result);
}

public static <T> Gatherer<T,?,T> deduplicateAdjacent() {  no usages  new *
    class State { T prev; boolean hasPrev; }  no usages  new *
    return Gatherer.of(
            State::new,
            (state, element, downstream) -> {
                if (!state.hasPrev) {
                    state.hasPrev = true;
                    state.prev = element;
                    return downstream.flush(element);
                } else if (!Objects.equals(state.prev, element)) {
                    state.prev = element;
                    return downstream.flush(element);
                } else {
                    return true; // skip duplicate
                }
            },
            Gatherer.unsupportedCombiner(),
            (state, downstream) -> {}
    );
}
```

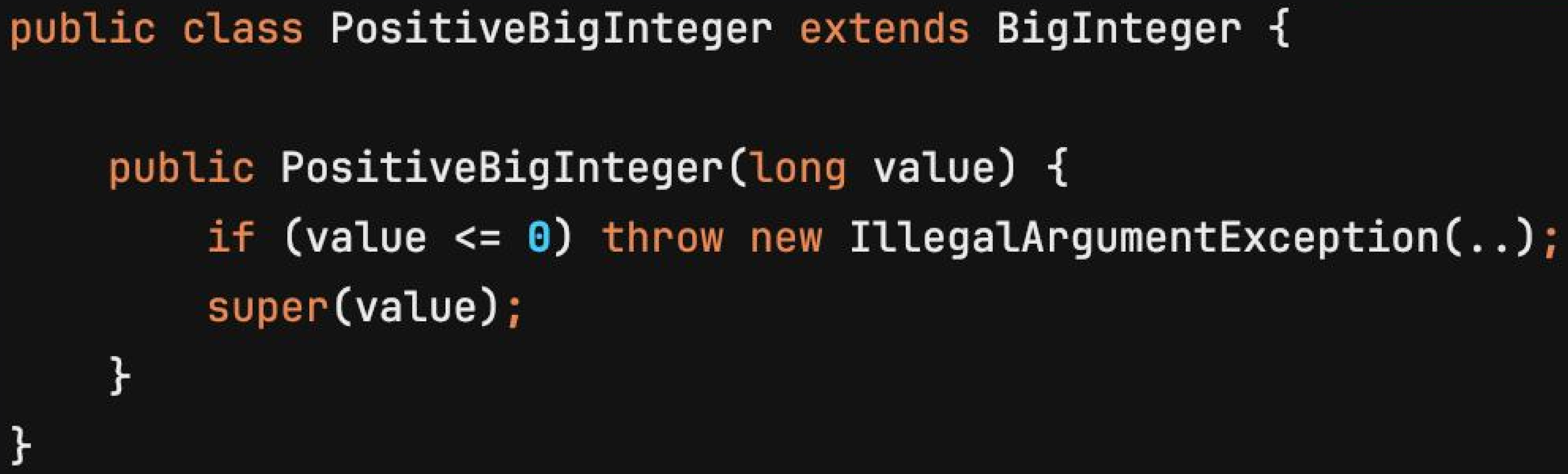
JEP 473: Stream Gatherers (Second Preview)

8

JEP 482

Flexible Constructor Bodies

Второе превью



```java
public class PositiveBigInteger extends BigInteger {

    public PositiveBigInteger(long value) {
        if (value <= 0) throw new IllegalArgumentException(..);
        super(value);
    }
}
```

In constructors in the Java programming language, allow statements to appear before an explicit constructor invocation, i.e., super(..) or this(..).

The statements cannot reference the instance under construction, but they can initialize its fields. Initializing fields before invoking another constructor makes a class more reliable when methods are overridden

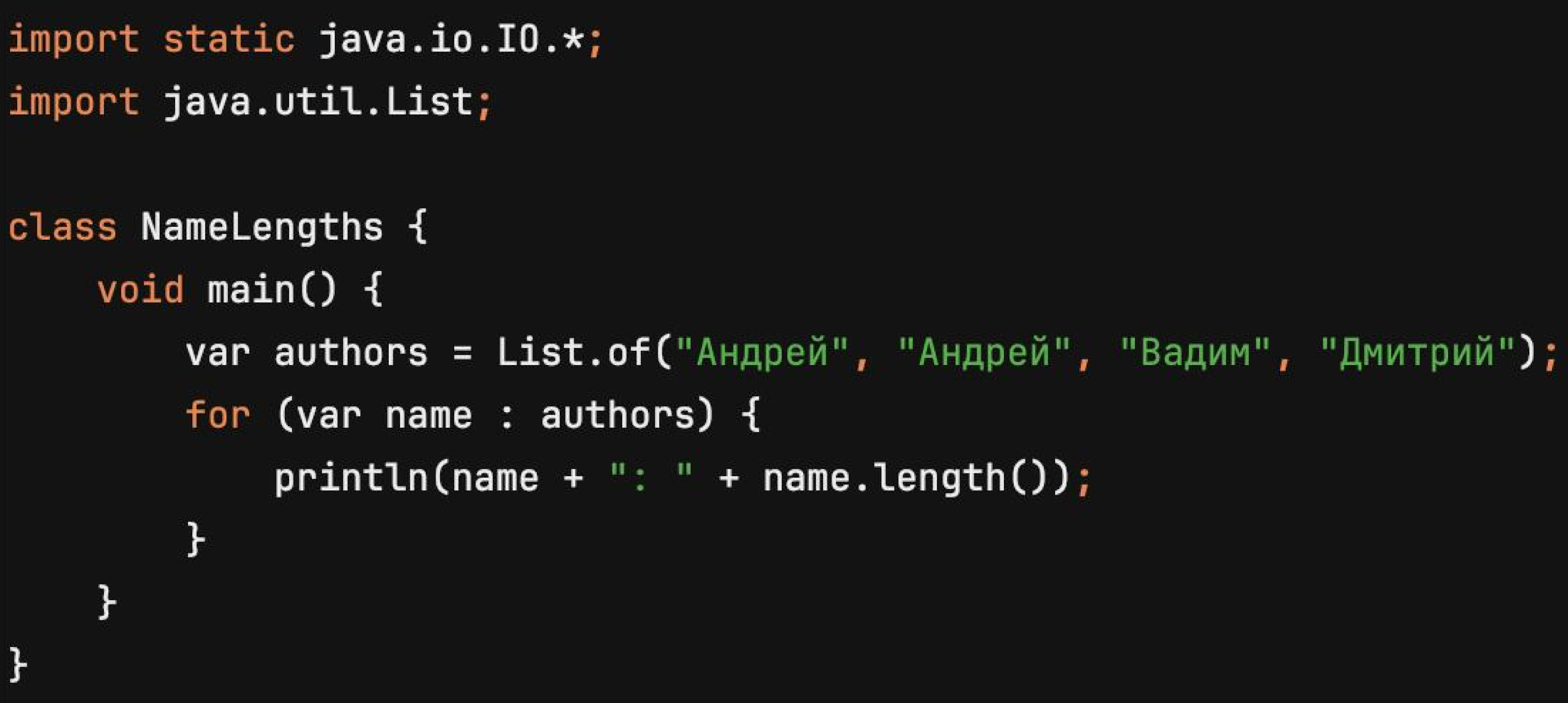
9

JEP 482

Implicitly Declared Classes and Instance Main Methods

Третье превью

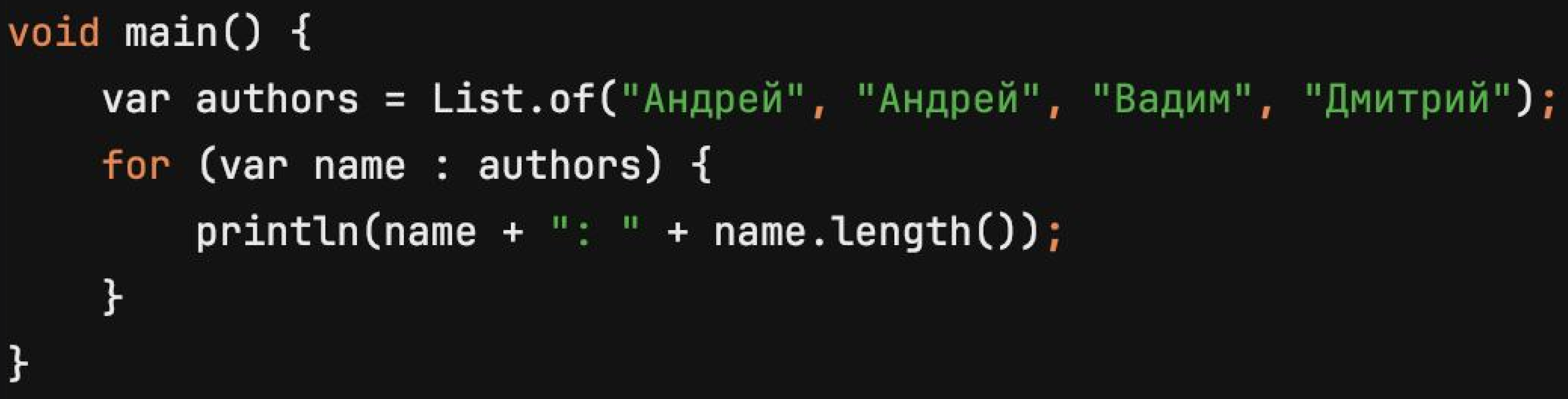
Было



```
import static java.io.IO.*;
import java.util.List;

class NameLengths {
    void main() {
        var authors = List.of("Андрей", "Андрей", "Вадим", "Дмитрий");
        for (var name : authors) {
            println(name + ": " + name.length());
        }
    }
}
```

JEP 477: Implicitly Declared Classes and Instance Main Methods

Стало



```java
void main() {
    var authors = List.of("Андрей", "Андрей", "Вадим", "Дмитрий");
    for (var name : authors) {
        println(name + ": " + name.length());
    }
}
```

JEP 477: Implicitly Declared Classes and Instance Main Methods

10

JEP 481

Scoped Values

Третье превью

```java
private static final ScopedValue<String> X = ScopedValue.newInstance();

void foo() {
    where(X, "hello").run(() -> bar());
}

void bar() {
    System.out.println(X.get()); // prints hello
    where(X, "goodbye").run(() -> baz());
    System.out.println(X.get()); // prints hello
}

void baz() {
    System.out.println(X.get()); // prints goodbye
}
```

11

JEP 480

Structured Concurrency

Третье превью без изменений

```
@Override
public Response handle(Request request, Response response) {
    try (var scope = new StructuredTaskScope.ShutdownOnFailure()) {
        Supplier<UserInfo>    user   = scope.fork(() -> readUserInfo());  // (1)
        Supplier<List<Offer>> offers = scope.fork(() -> fetchOffers());   // (2)
        scope.join().throwIfFailed();  // Wait for both forks
        return new Response(user.get(), order.get());
    } catch (Exception ex) {
        reportError(response, ex);
    }
}
```

12

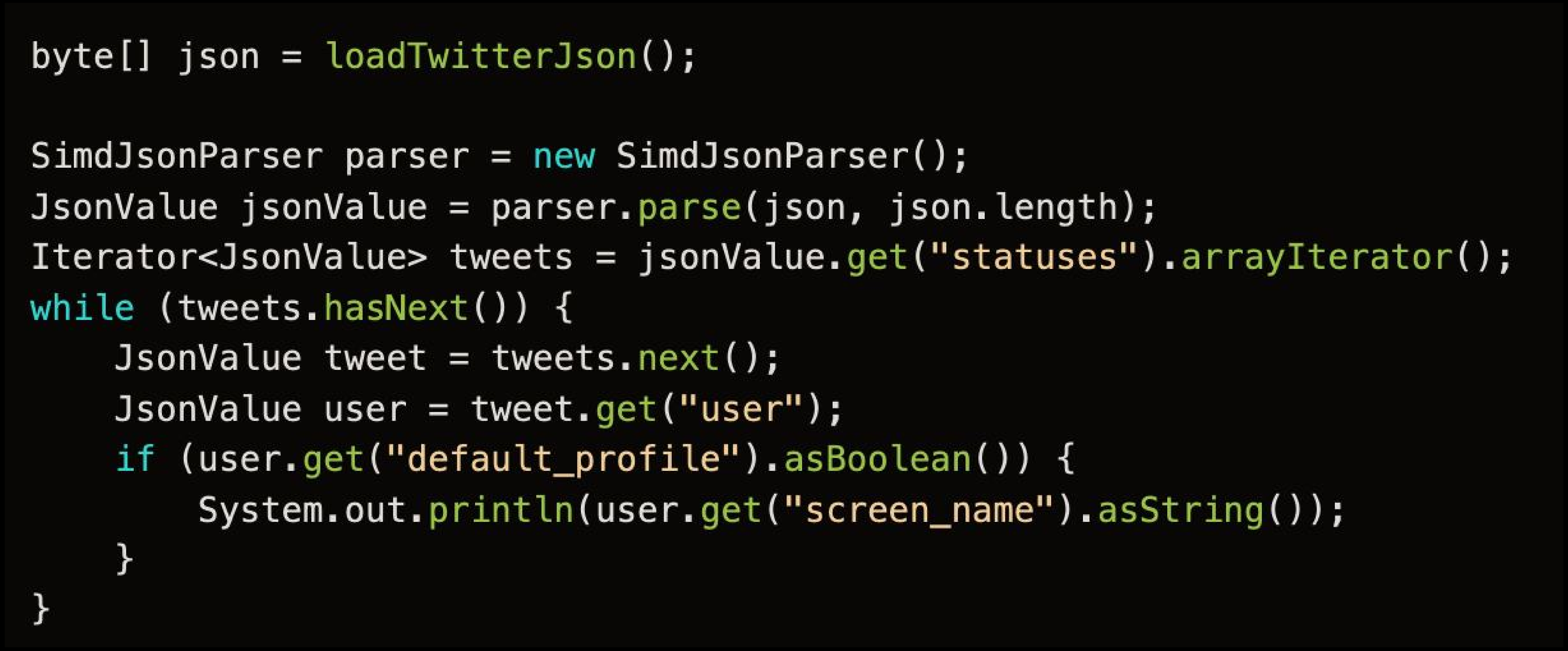
JEP 469

Vector API

В восьмом инкубаторе

simdjson-java

A Java version of simdjson - a JSON parser using SIMD instructions, based on the paper Parsing Gigabytes of JSON per Second by Geoff Langdale and Daniel Lemire.



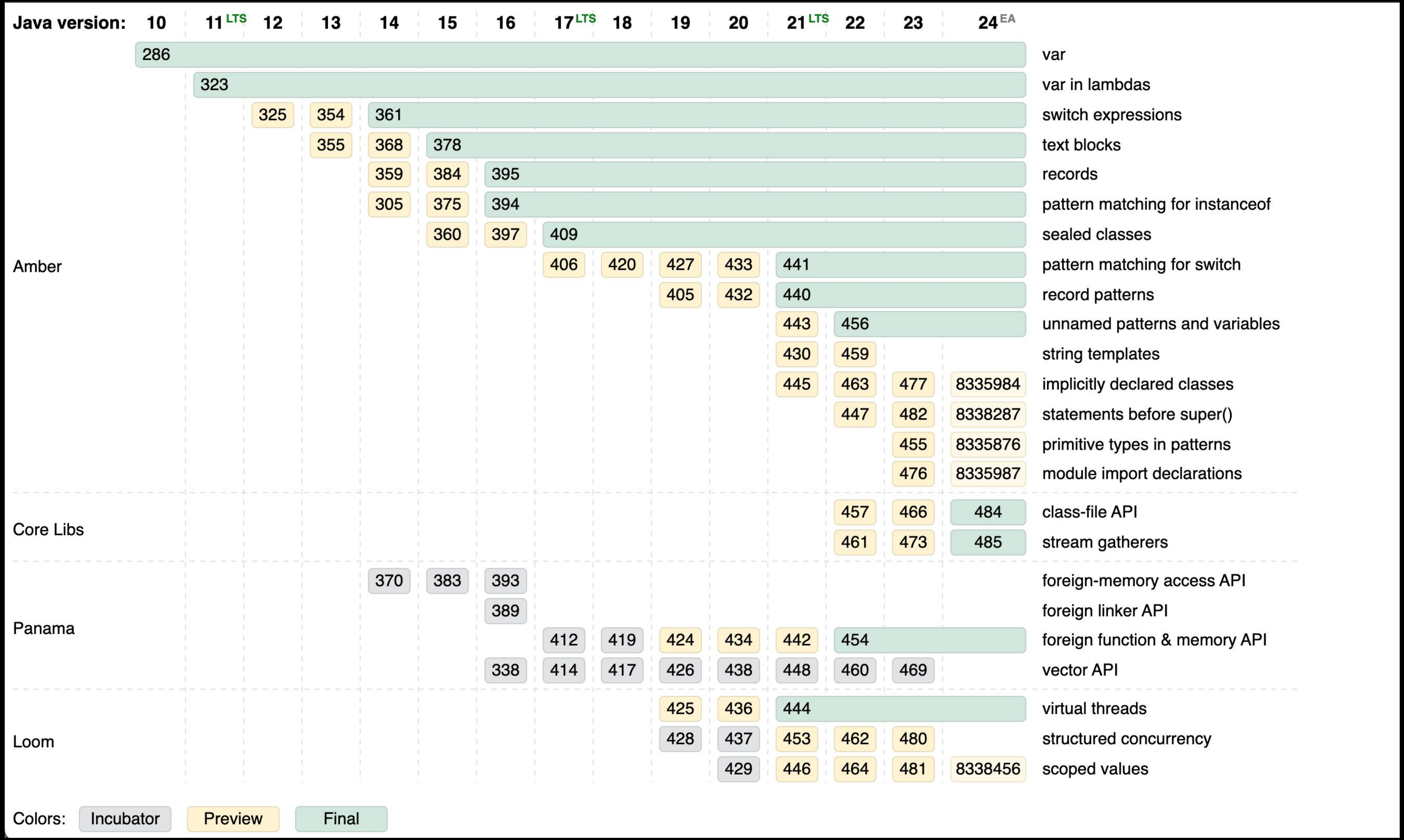
```java
byte[] json = loadTwitterJson();

SimdJsonParser parser = new SimdJsonParser();
JsonValue jsonValue = parser.parse(json, json.length);
Iterator<JsonValue> tweets = jsonValue.get("statuses").arrayIterator();
while (tweets.hasNext()) {
    JsonValue tweet = tweets.next();
    JsonValue user = tweet.get("user");
    if (user.get("default_profile").asBoolean()) {
        System.out.println(user.get("screen_name").asString());
    }
}
```

Спасибо!

Дмитрий Волыхин

@Javaswag

Java 24



Java version:	10	11 LTS	12	13	14	15	16	17 LTS	18	19	20	21 LTS	22	23	24 EA	
Amber	286															var
		323														var in lambdas
			325	354	361											switch expressions
				355	368	378										text blocks
					359	384	395									records
					305	375	394									pattern matching for instanceof
						360	397	409								sealed classes
								406	420	427	433	441				pattern matching for switch
										405	432	440				record patterns
												443	456			unnamed patterns and variables
												430	459			string templates
												445	463	477	8335984	implicitly declared classes
													447	482	8338287	statements before super()
														455	8335876	primitive types in patterns
														476	8335987	module import declarations
Core Libs													457	466	484	class-file API
													461	473	485	stream gatherers
Panama					370	383	393									foreign-memory access API
							389									foreign linker API
								412	419	424	434	442	454			foreign function & memory API
							338	414	417	426	438	448	460	469		vector API
Loom										425	436	444				virtual threads
										428	437	453	462	480		structured concurrency
											429	446	464	481	8338456	scoped values

Colors: Incubator Preview Final

Вот теперь точно

Спасибо!