



Java wird 30

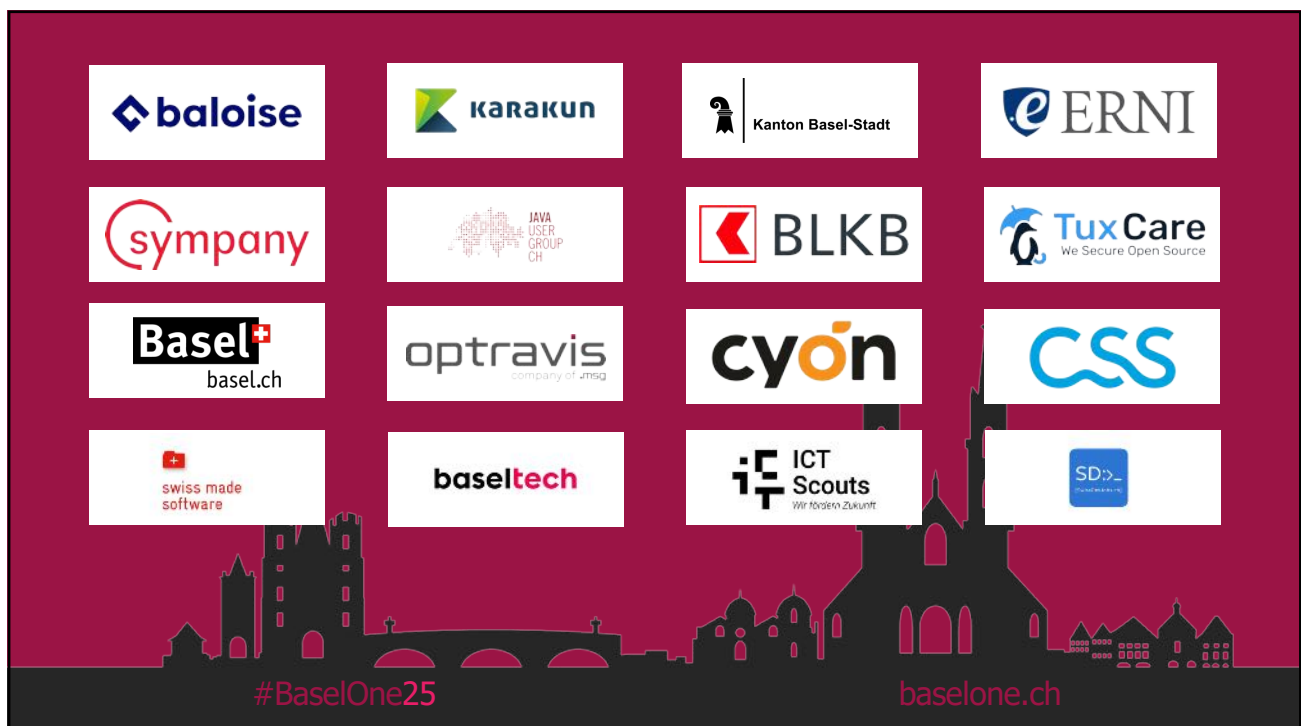
Die Neuerungen in Java 21 bis 25

Falk Sippach

16.10.2025

embarc 

0



1



Abstract

Die Neuerungen in Java 21 bis 25

Alle halben Jahre erscheinen Major-Releases mit einer Vielzahl von neuen Features. Manche starten als Inkubator und viele durchlaufen mehrere Preview-Phasen. Und auch wenn wir die neuen Funktionen nicht direkt einsetzen können, lohnt sich immer der Blick auf den aktuellsten Stand. Aktuell geht es um so spannende Themen wie Erweiterungen zum Pattern Matching, Bibliotheken rund um Virtual Threads, String Templates, Stream Gatherers, Statements before super, Compact Source Files and Instance Main Methods, die Class File API und Stable Values.

Neben diesen verschiedenen JDK Enhancement Proposals (JEPs) werfen wir natürlich auch einen Blick auf hilfreiche API-Verbesserungen und Änderungen an der JVM, z. B. bei den Garbage Collectoren. Ihr bekommt einen Überblick über die neusten Entwicklungen im Java-Umfeld und seht heute schon, was Euch in den nächsten Jahren in der täglichen Arbeit erwarten wird.



Falk Sippach

- Softwarearchitekt, Berater, Trainer bei embarc
- früher bei Orientation in Objects (OIO), Trivadis

Schwerpunkte

- Architekturberatung und -bewertung
- Cloud- und Java-Technologien





Teil der Java Community



<https://www.jug-da.de/>

Nächster Termin am 23.10.



<https://cyberland.ijug.eu/>

monatliche Opensource
Code Camps

Agenda

Was euch erwartet

Die neuen Features

Fazit + Ausblick

Java - still a thing?

01

02

Was ist sonst passiert?

03

04

01.

Halbjährlicher Release-Prozess

Seit über 5 Jahren wird halbjährlich eine neue Version des OpenJDK herausgebracht. Wie läuft der Prozess?



7

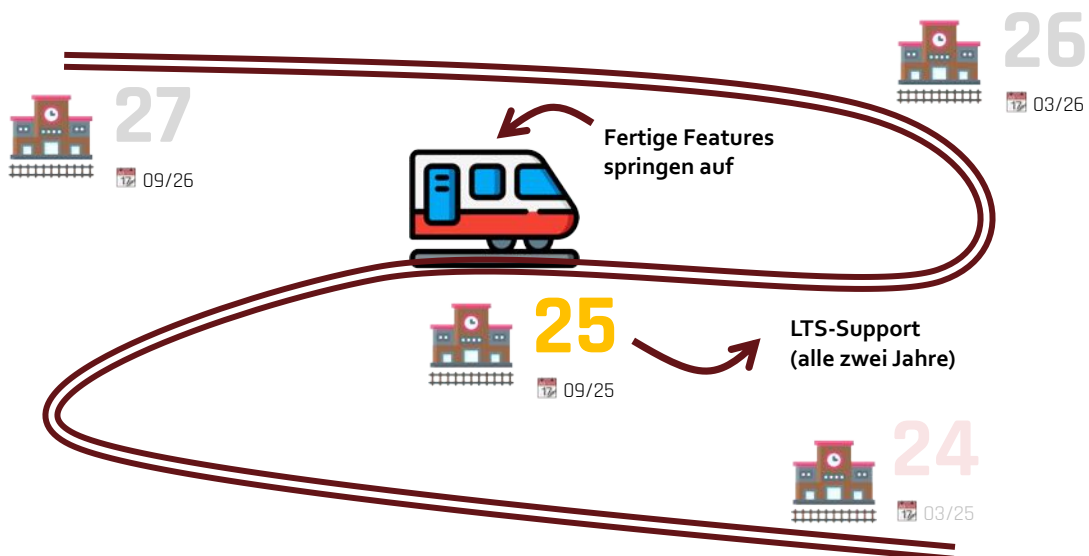

embarc.de
Die Neuerungen in Java 21 bis 25
8

Warum ist Java auch nach 30 Jahren immer **noch so populär?**

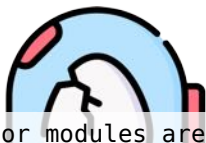
8



Halbjährlicher Release-Train



Inkubator und Preview



Incubator modules are a means of putting non-final APIs and non-final tools in the hands of developers, while the APIs/tools progress towards either finalization or removal in a future release.

<https://openjdk.org/jeps/11>



A *preview feature* is a new feature [...] that is fully specified, fully implemented, and yet impermanent. It is available in a JDK feature release to provoke developer feedback based on real world use; this may lead to it becoming permanent in a future Java SE Platform.

<https://openjdk.org/jeps/12>



Java 24

Build

Client Libraries

Compatibility & Specification

Review

Compiler

Conformance

Core Libraries

Governing Board

HotSpot

IDE Tooling & Support

Internationalization

JMX

Members

Networking

Porters

Quality

Security

Serviceability

Vulnerability

Web

Projects

(overview, archive)

Amber

Babylon

Chac

Code Tools

Coin

Common VM

Interface

Developers' Guide

Device I/O

Duke

Galahad

Grail

IcedTea

JDK 7

JDK 8

JDK 8 Updates

JDK 9

JDK 10

JDK 11

JDK 12

JDK 13

JDK 14

JDK 15

JDK 16

JDK 17

JDK 18

JDK 19

JDK 20

JDK 21

JDK 22

JDK 23

JDK 24

JDK 25

JDK Updates

Jigsaw

Kona

Kulla

Lanai

Leyden

Lilliput

Locale Enhancement

Loom

Memory Model

Update

Metropolis

Mission Control

Multi-Language VM

Nashorn

New I/O

ProjectFX

Schedule

2024/12/05

Rampdown Phase One (branch from main line)

2025/01/16

Rampdown Phase Two

2025/02/06

Initial Release Candidate

2025/02/20

Final Release Candidate

2025/03/18

General Availability

Features

404: Generational Shenandoah (Experimental)

450: Compact Object Headers (Experimental)

472: Prepare to Restrict the Use of JNI

475: Late Barrier Expansion for G1

478: Key Derivation Function API (Preview)

479: Remove the Windows 32-bit x86 Port

483: Ahead-of-Time Class Loading & Linking

484: Class-File API

485: Stream Gatherers

486: Permanently Disable the Security Manager

487: Scoped Values (Fourth Preview)

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

489: Vector API (Ninth Incubator)

490: ZGC: Remove the Non-Generational Mode

491: Synchronize Virtual Threads without Pinning

492: Flexible Constructor Bodies (Third Preview)

493: Linking Run-Time Images without JMODs

494: Module Import Declarations (Second Preview)

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

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

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

498: Warn upon Use of Memory-Access Methods in sun.misc.Unsafe

499: Structured Concurrency (Fourth Preview)

501: Deprecate the 32-bit x86 Port for Removal

der Umstel

halbjährliche

24 J

Neuer Rekord seit der Umstellung auf halbjährliche Releases:

24 JEPs

18



Java 25

OpenJDK	JDK 25
Installing	This release will be the Reference Implementation of version 25 of the Java Platform, as specified by JSR 400 in the Java Community Process.
Contributing	
Sponsoring	
Developers' Guide	
Vulnerabilities	
JDK GAVIA Builds	
Mailing Lists	
Wiki - IRC	
Mastodon	
Bluesky	
Bylaws - Census	
Legal	
Workshop	
JEP Process	
Source code	
GitHub	
Mercurial	
Tools	
Git	
Trivy 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	
Security	
Serviceability	
Vulnerability	
Web	
Projects	
(overview, archive)	
Amber	
Babylon	
Chac	
Code Tools	
Coin	
Common VM	
Interface	
Developers' Guide	
ProjectFX	

Schedule	
2025/06/05	Rampdown Phase One (branch from main line)
2025/07/17	Rampdown Phase Two
2025/08/07	Initial Release Candidate
2025/08/21	Final Release Candidate
2025/09/16	General Availability

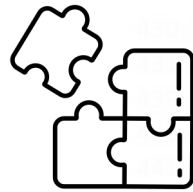
Features	
470:	PEM Encodings of Cryptographic Objects (Preview)
502:	Stable Values (Preview)
503:	Remove the 32-bit x86 Port
505:	Structured Concurrency (Fifth Preview)
506:	Scoped Values
507:	Primitive Types in Patterns, Instanceof, and switch (Third Preview)
508:	Vector API (Tenth Incubator)
509:	JFR CPU-Time Profiling (Experimental)
510:	Key Derivation Function API
511:	Module Import Declarations
512:	Compact Source Files and Instance Main Methods
513:	Flexible Constructor Bodies
514:	Ahead-of-Time Command-Line Ergonomics
515:	Ahead-of-Time Method Profiling
518:	JFR Cooperative Sampling
519:	Compact Object Headers
520:	JFR Method Timing & Tracing
521:	Generational Shenandoah

Last update: 2025/8/7 22:51 UTC

immerhin wieder:

18 JEPs

19



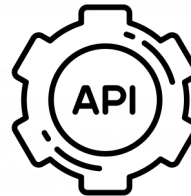
Neue Features



Garbage Collectoren



Interna



API-Änderungen (JDK)

02.

Die neuen Features

Was ist überraschend, was ist besonders relevant?



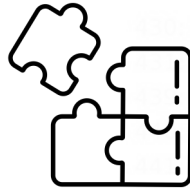


embarc.de

Die Neuerungen in Java 21 bis 25

23

Features



Neue Features



Garbage Collectoren



Interna



API-Änderungen (JDK)

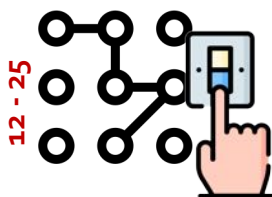
23



embarc.de

Die Neuerungen in Java 21 bis 25

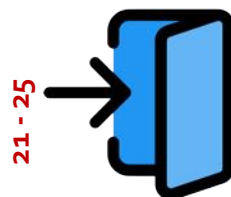
27



Pattern Matching



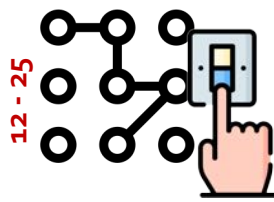
Rund um Virtual
Threads



Paving the on-ramp



27



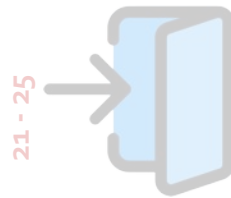
12 - 25

Pattern Matching



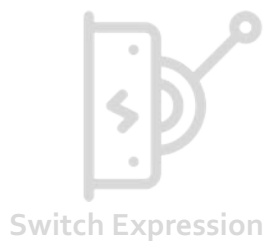
19 - 25

Rund um Virtual
Threads

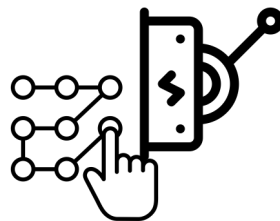


21 - 25

Paving the on-ramp



Switch Expression



Pattern Matching for
switch



Record Patterns
(Deconstruction)



Type Patterns
(Pattern Matching
for instanceof)



Records



Unnamed Patterns
and Variables



Primitive
Type Patterns



Sealed Classes





Pattern Matching



„Pattern matching is a mechanism for **checking a value against a pattern**. A successful match can also **deconstruct a value into its constituent parts**.

It is a more **powerful version of the switch statement in Java** and it can likewise be used in place of a series of if/else statements.“



<https://docs.scala-lang.org/tour/pattern-matching.html>



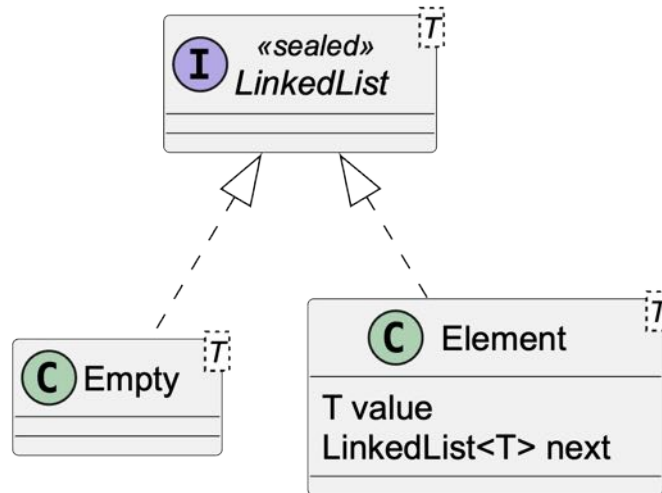
Neues Programmierparadigma?



<https://www.infoq.com/articles/data-oriented-programming-java/>



Algebraische Datentypen



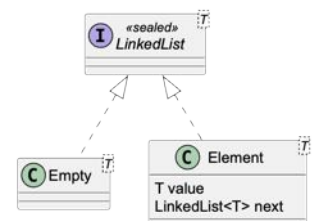
```

        Sealed Class/Interface
public sealed interface LinkedList<T>
    permits LinkedList.Element, LinkedList.Empty {

    Record record Element<T>(T value, LinkedList<T> next)
        implements LinkedList<T> {}

    final class Empty<T> implements LinkedList<T> {}

    static <T> LinkedList<T> of(T... values) {
        ..
    }
}
    
```

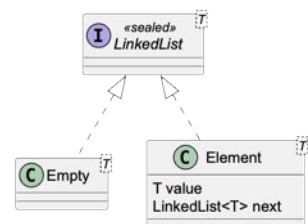




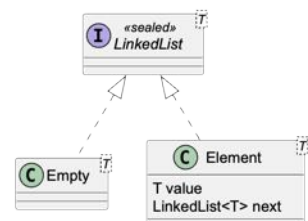
```
static <T> LinkedList<T> of(T... values) {
    if (values.length == 0) return new LinkedList.Empty<>();

    LinkedList<T> current = new LinkedList.Empty<>();

    for (int i = values.length - 1; i >= 0; i--) {
        current = new LinkedList.Element<>(values[i], current);
    }
    return current;
}
```



```
LinkedList<Integer> list = of(1, 2, 3);
System.out.println(contains(5, list)); // false
System.out.println(contains(1, list)); // true
```





switch-Expression

```
static <T> boolean contains(T value, LinkedList<T> list) {
    return switch (list) {
        case Empty empty -> false;
        case Element<T>(T v, var tail)
            when Objects.equals(v, value) -> true;
        case Element<T>(T v, var tail) -> contains(value, tail);
    };
}
```

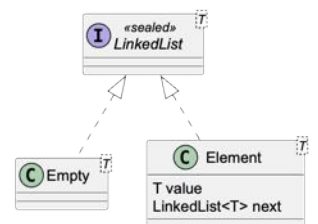
Type Pattern
(Pattern Matching for instanceof)

Record Pattern

when-Clause (Guarded Pattern)

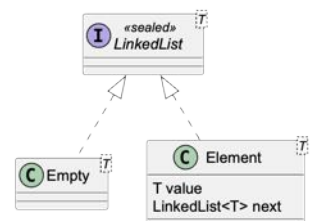


Kein Default notwendig (Exhaustiveness-Prüfung durch Compiler)!



```
static <T> boolean contains(T value, LinkedList<T> list) {
    return switch (list) {
        case Empty _ -> false;
        case Element<T>(T v, _)
            when Objects.equals(v, value) -> true;
        case Element<T>(_, var tail) -> contains(value, tail);
    };
}
```

Unnamed Pattern





Unnamed Variables



```
try {
    var result = 5 / 0;
} catch (ArithmeticException _) {
    System.out.println("Division nicht möglich");
}

System.out.println(new ArrayList<>(List.of(1, 2, 3))
    .stream()
    .map(_ -> 42)
    .toList()); // [42, 42, 42]
```



Primitive Type Patterns (1)



Primitive Patterns räumen drei Altlasten auf –

- Asymmetrie
- unnötiges Boxing/Handarbeit
- und Löcher bei Record-Deconstruction und switch.



Primitive Type Patterns (2)



```
static String level(int flights) {
    return switch (flights) {
        case 0 -> "New";
        case 1, 2 -> "Bronze";
        case int i when i >= 100 -> "Gold";    // Guard
        case int i -> "Silver (" + i + ')';
    };
}
```



Primitive Type Patterns (3)



```
int value = -128;
if (value instanceof byte b && b > 0) {
    System.out.println("b has a positive byte value: " + b);
} else {
    System.out.println("negative or not of type byte: " + value);
}
```

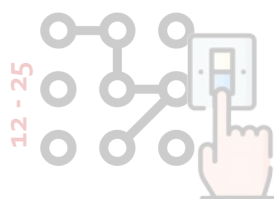


Primitive Type Patterns (4)



```
record PointWithDoubles(double x, double y) {}
record ColoredPoint(PointWithDoubles p, String color) {}

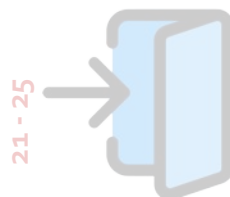
void describe(ColoredPoint cp) {
    switch (cp) {
        case ColoredPoint(PointWithDoubles(int x, int y), var color) ->
            System.out.printf("Pixel color %s at (%d, %d)%n", color, x, y);
        case ColoredPoint(PointWithDoubles(double x, double y), var color) ->
            System.out.printf("Precise color %s at (%.2f, %.2f)%n", color, x, y);
    }
}
```



Pattern Matching



Rund um Virtual
Threads



Paving the on-ramp





Project Loom

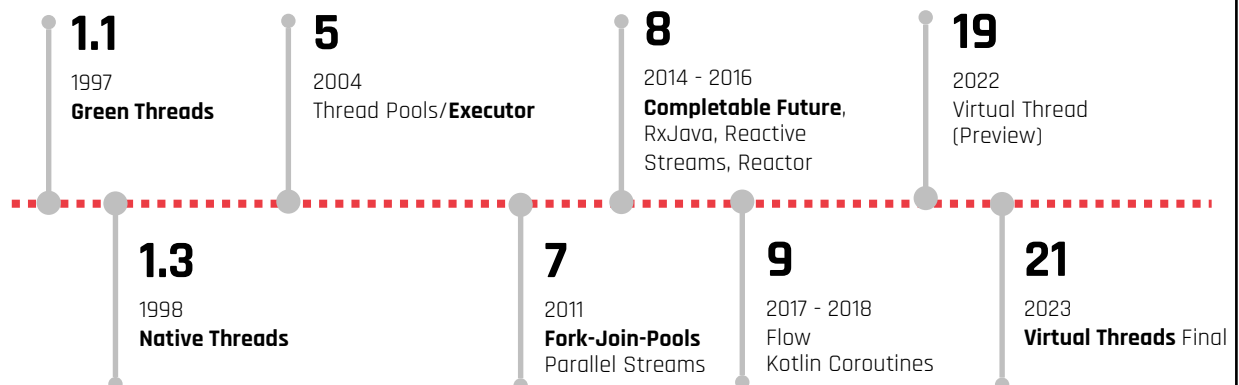


”

JVM features and APIs for supporting easy-to-use, high-throughput, lightweight concurrency and new programming models.



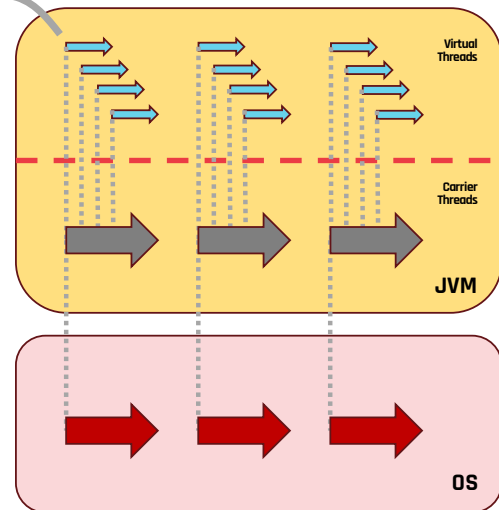
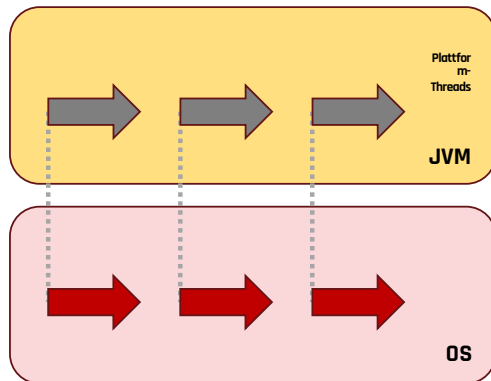
Die Geschichte von Threads in Java





Plattform- vs. Virtual Threads

Leichtgewicht, schnell erzeugt, geringer Ressourcenverbrauch



Millionen von Threads ...

```
Thread.Builder builder = Thread.ofVirtual();
// Thread.Builder threadBuilder = Thread.ofPlatform();

builder.start(() -> {
    // im Thread auszuführender Code
    System.out.println(
        Thread.currentThread().isVirtual());
});
```





Aber es geht gar nicht um Millionen von Threads



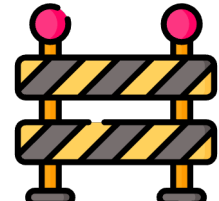
klassisches
Thread-API



Vereinfachter
Umgang mit vielen,
blockierenden Tasks
(IO)



nicht schneller oder
effizienter (laufen
auch nur auf
Platform Threads)



nicht für sehr
rechenintensive
(**lange blockierende**)
Tasks



Pinning bei synchronize



JEP 491: **Synchronize**
Virtual Threads
without Pinning

24



Für wen ist es gut?

viele/mehr parallele Requests

aber externe Ressourcen bleiben Bottleneck



Virtual Threads unter der Haube

Wir alle, die Webanwendungen bauen und davon indirekt profitieren.

nicht für die Verarbeitung großer Datenmengen bzw. langlaufende Berechnungen

"naive" Programmierung von Threads (einfacheres API, leichtgewichtig, kein Pooling)

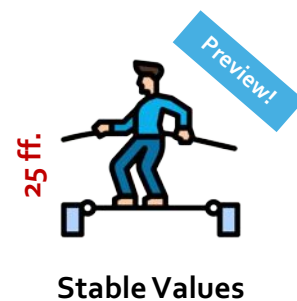
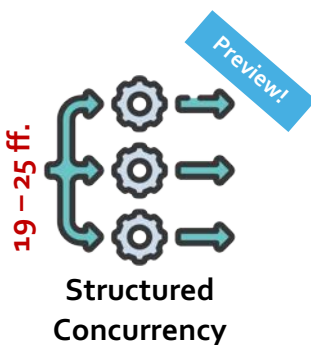


Nebenläufige Programmierung

Implementierung von sehr vielen und/oder blockierenden Aufgaben



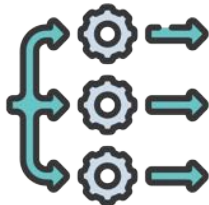
Weiteres im Umfeld von Virtual Threads



Funktioniert auch mit Platform Threads!



Structured Concurrency



Einfaches **Aufsplitten**
eine Haupt-Tasks in
Sub-Tasks



Behält Verbindung:
Main- und Sub-Task



Verringert Fehler
entstanden durch
**Annullierung und
Abschaltungen**



Passt sehr gut zu
Virtual Threads



Vor Structured Concurrency



```
private static final ExecutorService executor =  
    Executors.newCachedThreadPool();
```

```
Future<String> task1 = executor.submit() -> { ... }
```

```
Future<String> task2 = executor.submit() -> { ... }
```

```
Future<String> task3 = executor.submit() -> { ... }
```

```
System.out.println(task1.get());
```

```
System.out.println(task2.get());
```

```
System.out.println(task3.get());
```

Fragestellungen (Szenarien):

- 1. Thread gewinnt, restliche abbrechen
- alle abbrechen, wenn ein Fehler passiert
- alle müssen erfolgreich sein
- ...



Structured Concurrency



```
record Response(String user, int order) {}
```

alternative Strategien über:
Joiner (auch **eigene**
Implementierung)

```
Response handle() throws InterruptedException {
    try (var scope = StructuredTaskScope.<Object, Void>open()) {
        var user = scope.fork(this::findUser); // liefert String
        var order = scope.fork(this::fetchOrder); // liefert Integer
        scope.join(); // wartet auf alle
        return new Response((String) user.get(), (Integer) order.get());
    }
}
```

wartet auf alle, wirft ggf. **Exception** und
beendet die **anderen Tasks**

```
// Platzhalter:
String findUser() { return "alice"; }
int fetchOrder() { return 42; }
```



19 - 25

basiert auf
Virtual Threads



Structured Concurrency - Strategien



```
// Erstes erfolgreiches Ergebnis gewinnt
StructuredTaskScope.open(Joiner.<T>anySuccessfulResultOrThrow(),
    cfg -> cfg.withTimeout(java.time.Duration.ofSeconds(2)))
```

```
// Alle oder keiner
StructuredTaskScope.open(Joiner.<T>allSuccessfulOrThrow())
```

```
// Erfolgreiche und fehlgeschlagene Tasks unterscheiden
StructuredTaskScope.open(Joiner.<T>awaitAll())
```

```
// Früher Abbruch per Prädikat
StructuredTaskScope.open(
    Joiner.<String>allUntil(st ->
        st.state() == Subtask.State.SUCCESS
        && st.get().contains("OK")))
```

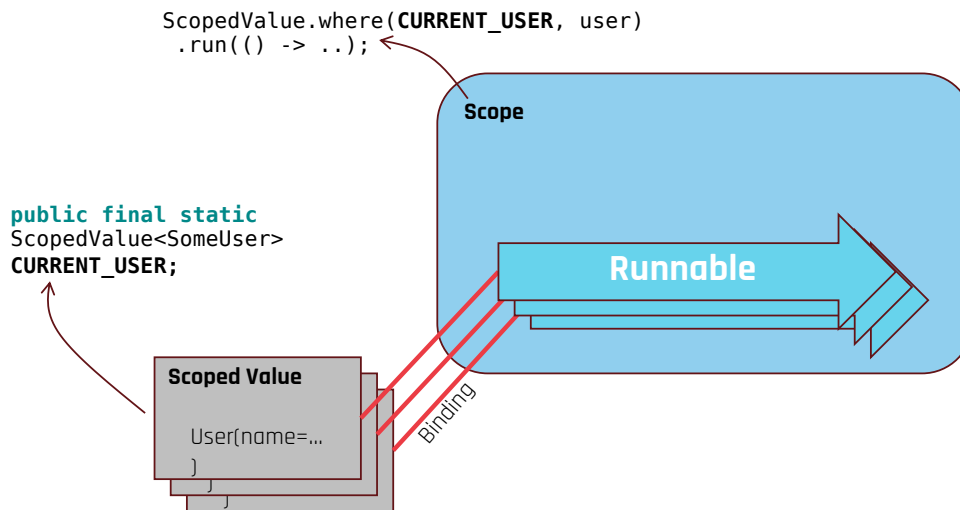


19 - 25

basiert auf
Virtual Threads



Scoped Values - Alternative zu ThreadLocal



Scoped Values



```

public final static ScopedValue<SomeUser> CURRENT_USER
    = ScopedValue.newInstance();

public void someControllerAction(HttpServletRequest request) {
    SomeUser user = authenticate(request);
    ScopedValue.where(CURRENT_USER, user)
        .run(() -> someService.processService());
    someService.processService();
}

void processService() {
    System.out.println(CURRENT_USER
        .orElseThrow(() -> new RuntimeException(".."))); }
    
```



Stable Values

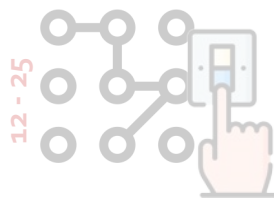
- "Immutability on demand" bzw. "Deferred Immutability"
- Ziele
 - schnellerer Start
 - Entkopplung von Erzeugung und Initialisierung
 - garantierte At-Most-One-Initialisierung in parallelen Szenarien
 - Konstanten-Optimierung durch JIT



Beispiel Stable Value

```
class OrderController {
    private final StableValue<Logger> logger
        = StableValue.of();

    private Logger logger() {
        return logger.orElseSet(() ->
            Logger.create(OrderController.class));
    }
}
```

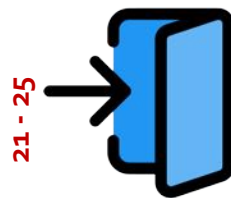



12 - 25

Pattern Matching



19 - 25

Rund um Virtual
Threads


21 - 25

Paving the on-ramp



Compact Source Files and Instance Main Methods



```
void main() {
    IO.println("Hello World")
}
```

wird zu

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

In Kombination mit **Launch Single-File Source-Code Programs** (JEP 330)

```
shell> java Main.java
```



Launch-Protokoll



```
protected static void main() {
    System.out.println("protected static void main()");
}

public void main(String[] args) {
    System.out.println("public void main()");
}
```



Gewinner, da
String[] args
als Parameter



Launch Multi-File Source-Code Programs

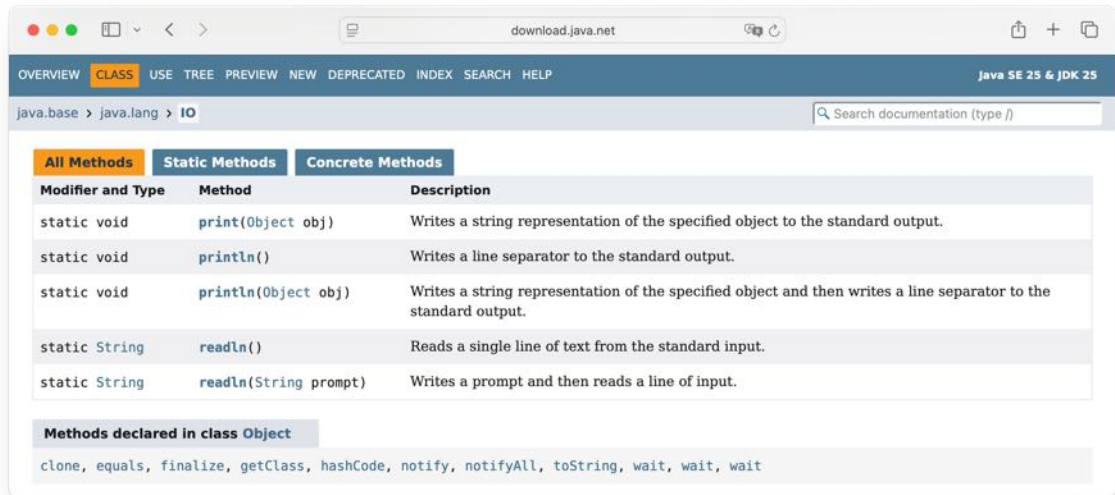
- Erweiterung des Single-File Source-Code
- nun beliebig viele Java-Dateien erlaubt
- JARs einbindbar
- Klassen im Package müssen entsprechend in Unterverzeichnissen liegen

```
StartWithJava
├─ Main.java
├─ Helper.java
├─ libs
└─ library.jar
```

```
> java -cp 'libs/*'
Main.java
```



java.lang.IO



OVERVIEW CLASS USE TREE PREVIEW NEW DEPRECATED INDEX SEARCH HELP		
java.base > java.lang > IO		
Search documentation (type /)		
All Methods	Static Methods	Concrete Methods
Modifier and Type	Method	Description
static void	print(Object obj)	Writes a string representation of the specified object to the standard output.
static void	println()	Writes a line separator to the standard output.
static void	println(Object obj)	Writes a string representation of the specified object and then writes a line separator to the standard output.
static String	readln()	Reads a single line of text from the standard input.
static String	readln(String prompt)	Writes a prompt and then reads a line of input.
Methods declared in class Object		
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait		



Module Imports

```
package de.sippsack.java25.modules;
```

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

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

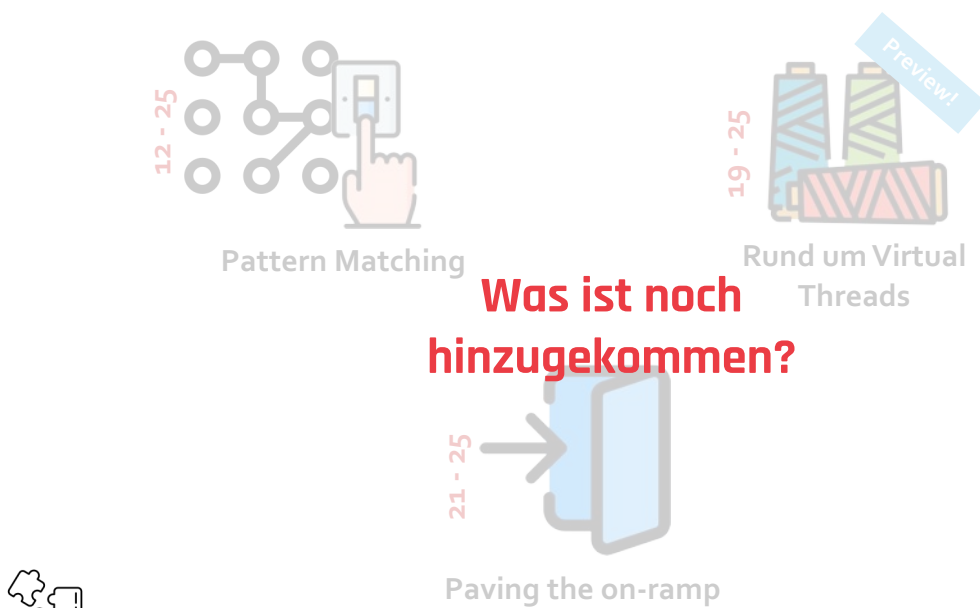
```
import java.util.Date;
```

```
public class ModuleImports {
```

```
    public static void main(String[] args) {
```

```
        System.out.println(new Date());
```

```
    }
```



Flexible Konstruktor Inhalte



```
public PositiveBigInteger(long value) {
    // Invalid before Java 22
    if (value <= 0) throw new
        IllegalArgumentException("non-positive value");
    super(value);
}
```



```
class Person {
    final String firstname;
    final String lastname;
    final LocalDate birthdate;

    public Person(String firstname, String lastname, LocalDate birthdate) {
        ...
    }
}

class Employee extends Person {
    private final String company;

    public Employee(String name, LocalDate birthdate, String company) {
        if (company == null || company.isEmpty()) {
            throw new IllegalArgumentException("company is null or empty");
        }
        String[] names = name.split("\\s");
        // System.out.println(this.company); // Zugriffe sind vor super() nicht erlaubt
        // System.out.println(this.firstname); // Zugriffe auf Felder der Oberklasse nicht erlaubt
        // promote(); // Instanzmethoden dürfen nicht vor super() aufgerufen werden
        super(names[0], names[1], birthdate);
        this.company = company;
    }
}
```



Klassisches Javadoc

```
/**
 * Returns the ... of {@code int} and {@code java.lang.String} arguments.
 * <p>
 * Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam
 * nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam
 * erat, sed diam voluptua.
 *
 * <p>
 * At vero eos et accusam et justo duo dolores et ea rebum.
 * <ul>
 * <li>foo</li>
 * <li>bar</li>
 * </ul>
 * <p>
 * For examples, see {@link #higherThanB(int, java.lang.String)}.
 *
 * @param x the first number as {@code int}
 * @param y the second number as {@code java.lang.String}
 * @return the higher value between {@code x} and {@code y} (converted to {@code int})
 * @throws NumberFormatException if {@code y} contains no number
 * @see #higherThanB(int, java.lang.String)
 * @since 7.5
 */
public static int higherThanA(int x, String y) {
    return 0;
}
```



Markdown in Javadoc

```

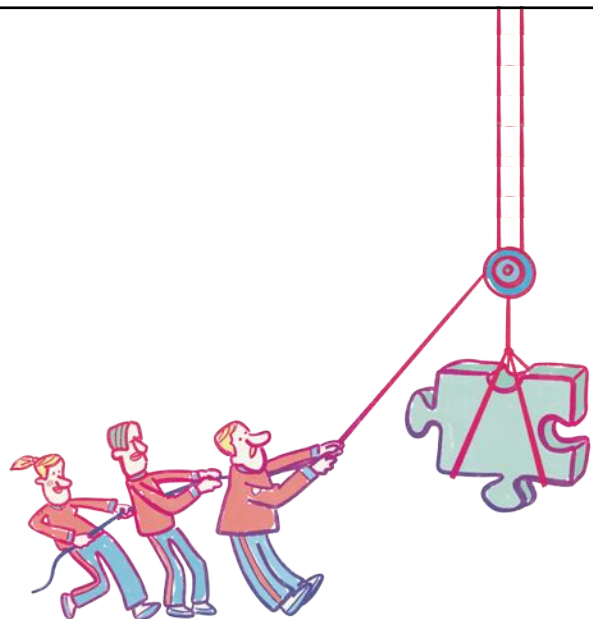
/// Returns the ... of 'int' and 'java.lang.String' arguments.
///
/// Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam
/// nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam
/// erat, sed diam voluptua.
///
/// At vero eos et accusam et justo duo dolores et ea rebum.
/// - foo
/// - bar
///
/// For examples, see \[ {@link }#higherThanB\(int, java.lang.String\)\].
///
/// @param x the first number as 'int'
/// @param y the second number as 'java.lang.String'
/// @return the higher value between 'x' and 'y' (converted to 'int')
/// @throws NumberFormatException if 'y' contains no number
/// @see #higherThanA(int, java.lang.String)
/// @since 7.5
public static int higherThanB(int x, String y) {
    return 0;
}

```

03.

Was ist sonst passiert?

Weitere Änderungen (intern, in der Klassenbibliothek, bei den Garbage Collectoren, ...)



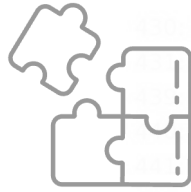


embarc.de

Die Neuerungen in Java 21 bis 25

92

Features



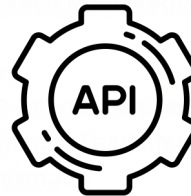
Neue Features



Garbage Collectoren



Interna



API-Änderungen (JDK)

92



embarc.de

Die Neuerungen in Java 21 bis 25

93

Features



Neue Features



Garbage Collectoren



Interna



API-Änderungen (JDK)

93



Neue Generation von Low-Latency Garbage Collectoren: ZGC und Shennadoah

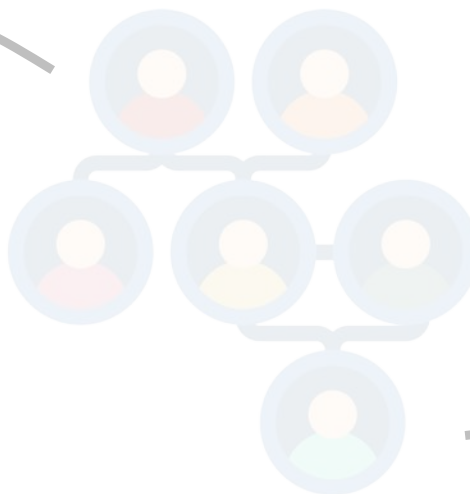
moderne Architekturen: Multi-Core + TB RAM

Ziel: kurze GC-Pausen im ms-Bereich



JEP 439: Generational ZGC

Old-Gen:
hat schon
mehrere GC-Läufe
überlebt, wird
länger leben



Young-Gen:
wird im GC-Lauf
prozessiert

JEP 474: ZGC: Generational Mode by Default



G1 ist weiterhin der
Default Garbage Collector



JEP 490: "ZGC: Remove the Non-Generational Mode"



new
24

 03/25

03/25

99 Die Neuerungen in Java 21 bis 25



API-Änderungen (JDK)

34



14 - 22



Foreign Linker + Foreign-Memory Access API

Finalisiert im
OpenJDK 22!

14 - 22 ff.



Vector API



Wartet auf die
Reformen des
Typsystems
(Value Classes)!



Class File API (Ersatz für ASM)

finalisiert in
Java 24

```
CodeModel code = ...
Set<ClassDesc> deps = new HashSet<>();
for (CodeElement e : code) {
    switch (e) {
        case FieldInstruction f -> deps.add(f.owner());
        case InvokeInstruction i -> deps.add(i.owner());
        // ... and so on for instanceof, cast, etc ...
    }
}
```

Einsatz von
Pattern Matching
für Analyse und
Modifizierung!



Class File API

```
CodeBuilder classBuilder = ...;
classBuilder.withMethod("fooBar",
    MethodTypeDesc.of(CD_void, CD_boolean, CD_int),
    flags,
    methodBuilder -> methodBuilder.withCode(codeBuilder -> {
        codeBuilder.iload(codeBuilder.parameterSlot(0))
        .ifThenElse(b1 -> b1.aload(codeBuilder.receiverSlot()))
        .iload(codeBuilder.parameterSlot(1))
        .invokevirtual(ClassDesc.of("Foo"),
            "foo",
            MethodTypeDesc.of(CD_void, CD_int)),
        b2 -> b2.aload(codeBuilder.receiverSlot()))
        .iload(codeBuilder.parameterSlot(1))
        .invokevirtual(ClassDesc.of("Foo"),
            "bar",
            MethodTypeDesc.of(CD_void, CD_int))
        .return_();
    })
);
```

```
void fooBar(boolean z, int x) {
    if (z)
        foo(x);
    else
        bar(x);
}
```



Methode fooBar erzeugen!



Class File API

```
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
                && mm.methodName()
                .stringValue().startsWith("debug"))) {
                classBuilder.with(ce);
            }
        }
    })
);
```

In bestehender Klasse alle debug-Methoden löschen!





JEP 450 - Compact Object Headers



Speicherverbrauch minimieren



Verkleinerung des Objekt-Headers
von 128 auf 64 Bit

new

24

03/25



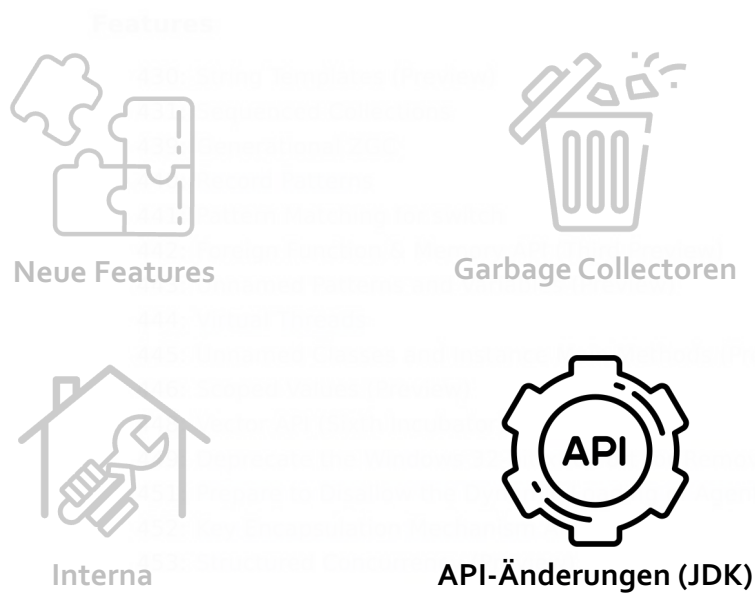
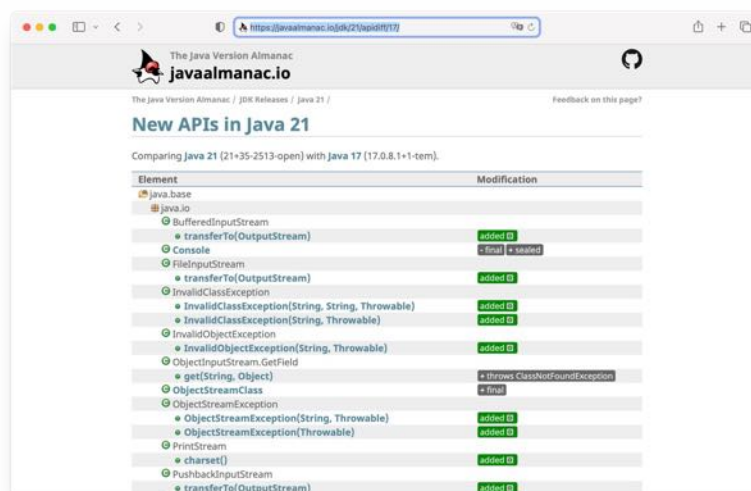
JEP 483: Ahead-of-Time Class Loading & Linking

- Verbesserung von Startzeit und Effizienz der Java Anwendungen
- häufig genutzte Klassen werden bereits zur Build-Zeit vorbereitet und nicht mehr erst zur Laufzeit geladen
- AOT-Cache-Lösung, in der Metadaten wie Konstanten, Methoden-Handles oder Lambda-Ausdrücke vorgeladen und gespeichert werden

new

24

03/25

Element	Modification
java base	
java.io	
BufferedInputStream	
transferTo(OutputStream)	added
Console	added
FileInputStream	
transferTo(OutputStream)	added
InvalidClassException	
InvalidClassException(String, String, Throwable)	added
InvalidClassException(String, Throwable)	added
InvalidObjectException	
InvalidObjectException(String, Throwable)	added
ObjectInputStream.GetField	
get(String, Object)	added
ObjectStreamClass	
ObjectStreamException	
ObjectStreamException(String, Throwable)	added
ObjectStreamException(Throwable)	added
PrintStream	
charset()	added
PushbackInputStream	
transferTo(OutputStream)	added

<https://javaalmanac.io/jdk/21/apidiff/17/>



Stream Gatherers



- Stream API liefert eine begrenzte Anzahl von Intermediate Operationen mit: **map**, **flatMap**, **filter**, ...
- immer wieder Wunsch nach weiteren Methoden
- jetzt ganze API zur Verfügung, um eigene Stream Gatherers zu schreiben



112



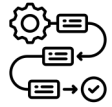
Analogie Collector API

- siehe <https://www.youtube.com/watch?v=jqUhObgDd5Q&t=2124s>
- verschiedene Erwartungen an Gatherers
- eigenen Vortrag draus machen

113



Beispiel: windowFixed



zustandsbehafteter N-N-Gatherer, der Eingabeelemente in Listen vorgegebener Größe gruppiert

```
// will contain: [[1, 2, 3], [4, 5, 6], [7]]
List<List<Integer>> windows = Stream.of(1, 2, 3, 4, 5, 6, 7)
    .gather(Gatherers.windowFixed(3))
    .toList();
System.out.println(windows);
```

Auspalten in
Gruppen von
je 3 Elementen!

114



Beispiel: fold



zustandsbehafteter N-1-Gatherer, baut ein Aggregat inkrementell auf und gibt dieses Aggregat am Ende aus

```
// will contain: Optional["12345"]
Optional<String> numberString =
    Stream.of(1, 2, 3, 4, 5)
        .gather(Gatherers.fold(() -> "",
            (string, number) -> string + number))
        .findFirst();
System.out.println(numberString);
```

Startwert: ""

Funktion:
Aufaddieren der
aktuellen Zahl auf
den String!

115



Weitere java.util.stream.Gatherers



- **mapConcurrent**: zustandsbehafteter 1-1-Gatherer, welcher die übergebene Funktion nebenläufig für jedes Input-Element aufruft
- **scan**: zustandsbehafteter 1-1-Gatherer, wendet eine Funktion auf dem aktuellen Zustand und dem aktuellen Element an, um das nächste Element zu erzeugen
- **windowSliding**: ähnlich zu windowFixed, nach dem ersten Rahmen wird der nächste Rahmen erzeugt, in dem das erste Element gelöscht wird und alle weiteren Werte nachrutschen

116



Eigene Gatherer



```
interface Gatherer<T, A, R> {
    default Supplier<A> initializer() {
        return defaultInitializer();
    };

```

Optional: Zustand über alle Stream-Elemente speichern.

```
    Integrator<A, T, R> integrator();

```

Muss implementiert werden: **integriert** die **Elemente** aus dem Eingabestrom.

```
    default BinaryOperator<A> combiner() {
        return defaultCombiner();
    }

```

Optional: Zusammenbringen einer parallelen Verarbeitung.

```
    default BiConsumer<A, Downstream<? super R>> finisher() {
        return defaultFinisher();
    };
    [...]
}

```

Optional: Aufruf am Ende aller Eingabe-elemente, kann auf den Zustand zugreifen.

117



JEP 496/497: Quantum-Resistant Module-Lattice-Based ...

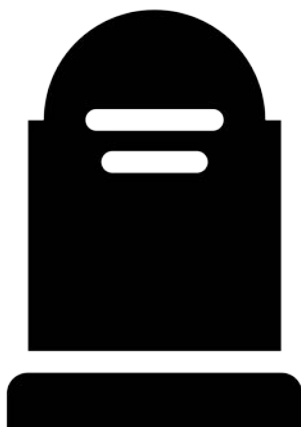
- ... Key Encapsulation Mechanism & Digital Signature Algorithm
- Algorithmen für **sichere Schlüsselaustauschverfahren** bzw. für **digitale Signaturen**, die vor zukünftige **Angriffe durch Quantencomputer** geschützt sind
- ML-KEM bzw. ML-DSA sind von NIST in FIPS 203 standardisierte Algorithmen

new
24

 03/25



Remove 32-Bit Ports



- JEP 479 (Remove the Windows 32-bit x86 Port)
- JEP 501 (Deprecate the 32-bit x86 Port for Removal)
- Architektur veraltet, keine neue 32-Bit Hardware, Betriebssystem-Support läuft aus
- Ziel: Ressourcen bei der Weiterentwicklung von Java effizienter nutzen

new
24

 03/25



Security

JEP 470 - PEM Encodings of Cryptographic Objects (Preview)

Stellt eine neue API zum Kodieren und Dekodieren von Schlüsseln, Zertifikaten und CRLs in das gängige PEM-Textformat vor:

PEMEncoder, PEMDecoder

JEP 510 - Key Derivation Function API

Macht die in JDK 24 eingeführte KDF-API (HKDF, Argon2 etc.) produktiv und ermöglicht eine einheitliche Verwendung von KDFs über `javax.crypto.KDF`.

new

25

 09/25


Java Flight Recorder

JEP 509 - JFR CPU-Time Profiling (Experimental)

JDK Flight Recorder kann auf Linux nun auch CPU-Zeit statt nur Wall-Clock-Zeit profilieren, was genauere Analysen CPU-gebundener Workloads erlaubt.

JEP 518 - JFR Cooperative Sampling

JFR erfasst Stack-Samples nun kooperativ an Safepoints, eliminiert riskante Heuristiken und verringert Absturz-Risiken bei gleichzeitig minimaler Safepoint-Bias.

JEP 520 - JFR Method Timing & Tracing

Ergänzt JFR um Bytecode-Instrumentation, um Aufrufhäufigkeit, Laufzeiten und optionale Stack-Traces für gezielt gefilterte Methoden exakt aufzuzeichnen.

new

25

 09/25



Performance

JEP 514 - Ahead-of-Time Command-Line Ergonomics

Ein neues `-XX:AOTCacheOutput` macht das Erstellen von AOT-Caches zum Ein-Schritt-Vorgang, indem Trainings- und Erstellphase automatisch verkettet werden.

JEP 515 - Ahead-of-Time Method Profiling

Speichert während des Trainingslaufes Methoden-Profile im AOT-Cache, sodass der JIT schon beim Produktionsstart mit optimierten Profilen kompiliert und die Aufwärmzeit sinkt.

new

25

 09/25

128

128

04.

Fazit und Ausblick

Zusammenfassung und Zukunft



129



Aktivierung Preview + Inkubator Features

```
$ javac --enable-preview -source 25 --add-modules
    jdk.incubator.concurrent StructuredConcurrency.java
```

```
$ java --enable-preview --add-modules
    jdk.incubator.concurrent StructuredConcurrency
```



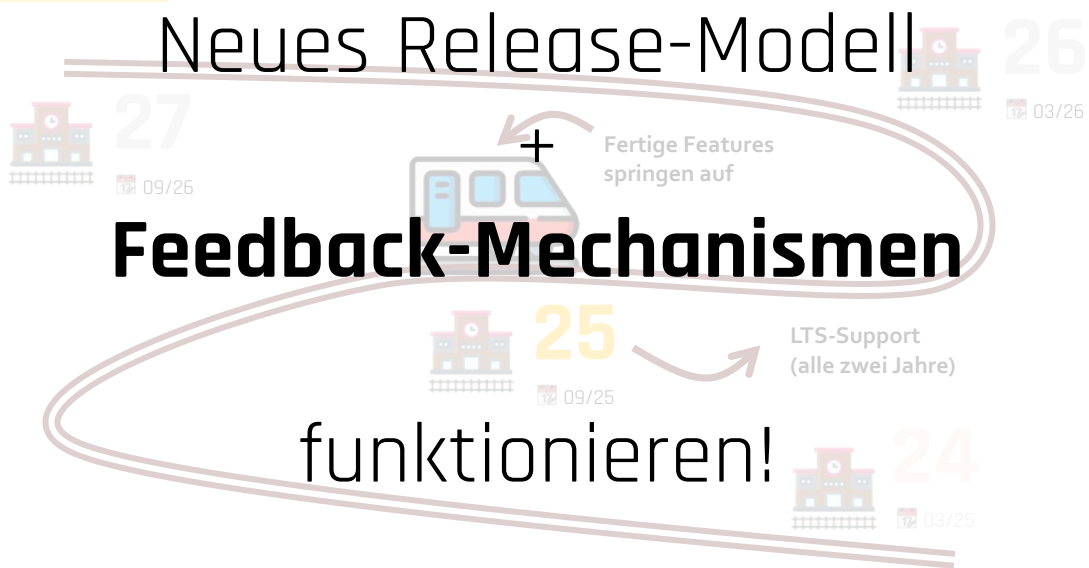
Die Java-Welt und
das Ökosystem sind
lebendig wie nie!



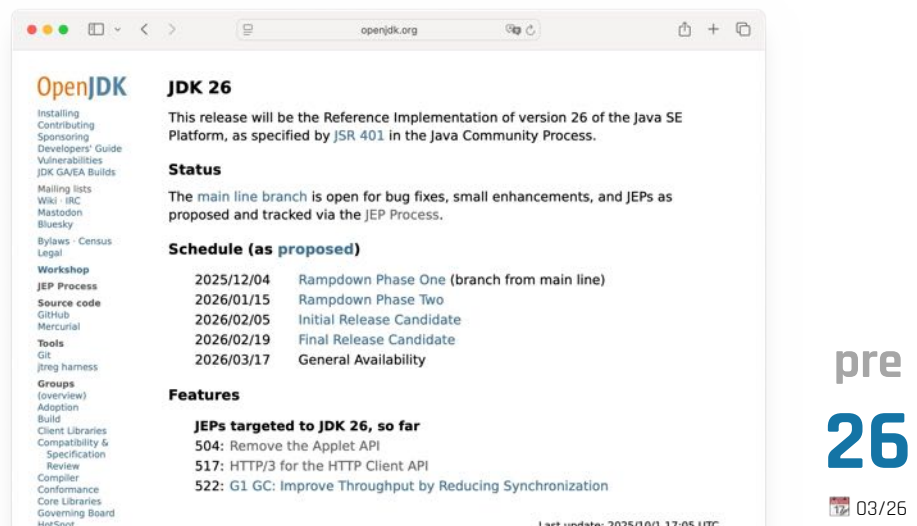


Halbjährlicher Release-Train

Neues Release-Modell



Java 26



OpenJDK

JDK 26

This release will be the Reference Implementation of version 26 of the Java SE Platform, as specified by JSR 401 in the Java Community Process.

Status

The main line branch is open for bug fixes, small enhancements, and JEPs as proposed and tracked via the JEP Process.

Schedule (as proposed)

2025/12/04	Rampdown Phase One (branch from main line)
2026/01/15	Rampdown Phase Two
2026/02/05	Initial Release Candidate
2026/02/19	Final Release Candidate
2026/03/17	General Availability

Features

JEPs targeted to JDK 26, so far

- 504: Remove the Applet API
- 517: HTTP/3 for the HTTP Client API
- 522: G1 GC: Improve Throughput by Reducing Synchronization

Last update: 2025/10/1 17:05 UTC

pre 26 03/26



embarc.de

Die Neuerungen in Java 21 bis 25

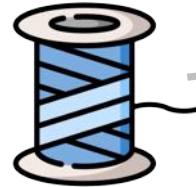
135

Pattern Matching +
kleine Syntax-
Verbesserungen



Amber

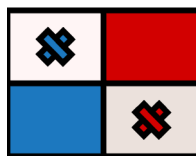
Virtual Threads



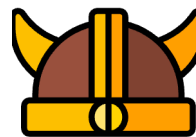
Loom

Was
kommt noch?

Vector API +
Foreign Function
& Memory API



Panama



Valhalla

Value Types

135



embarc.de

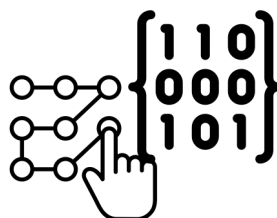
Die Neuerungen in Java 21 bis 25

136

Array Patterns



```
static void printFirstTwoStrings(Object o) {
    if (o instanceof String[] { String s1, String s2, ... }) {
        System.out.println(s1 + s2);
    }
}
```



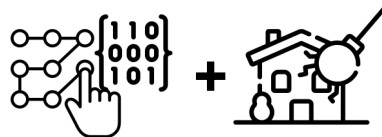
136



Kombination: Array & Records Patterns



```
static void printSumOfFirstTwoXCoords(Object o) {
    if (o instanceof Point[] {
        Point(var x1, var y1),
        Point(var x2, var y2),
        ... }) {
        System.out.println(x1 + x2);
    }
}
```



Record Patterns in foreach-Schleifen



MerlinBoe
@MBoegje

...

Latest news for our @Baselone workshop!! Awesome iteration with `for(Point(var x, var y) : points)` getting possible 🎉🎉🎉 #java #patterns

[Tweet übersetzen](#)



OpenJDK @OpenJDK · 19 Std.

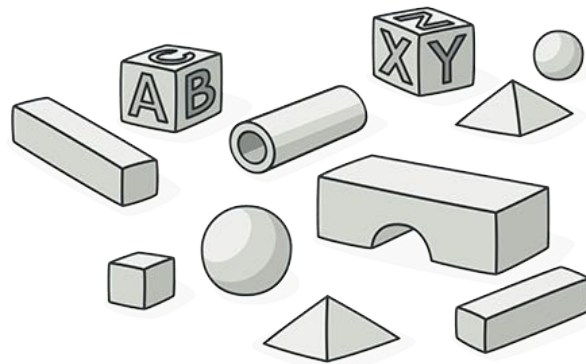
New candidate JEP: 432: Record Patterns (Second Preview): openjdk.org/jeps/432 #openjdk #java

8:18 vorm. · 19. Okt. 2022 · Twitter for iPhone





Primitive Obsession



<https://refactoring.guru/smells/primitive-obsession>



Project Valhalla



- "Advanced Java VM and Language feature candidates."

- Java kennt primitive Typen und Klassen
- Custom Types nur über Klassen
 - haben **Identität** und sind **Referenzen**
 - **Mutable**, extra **Memory** für Header, Locking/Synchronisierung, **Heap vs. Stack** (Speicher-Indirektion), **Nullable**



Nicht alle benutzer-definierten Typen brauchen das



Value Types



```
value class Point {
    private int x;
    private int y;

    public implicit Point();

    public Point(int x, int y) { .. }

    public Distance distanceTo(Point p) { .. }
}
Point! point = .. // can't be null
```

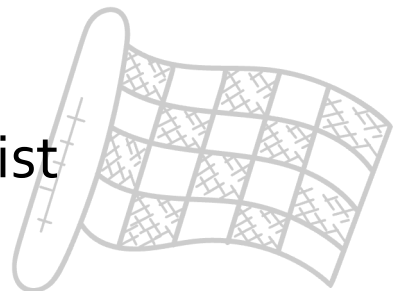


141

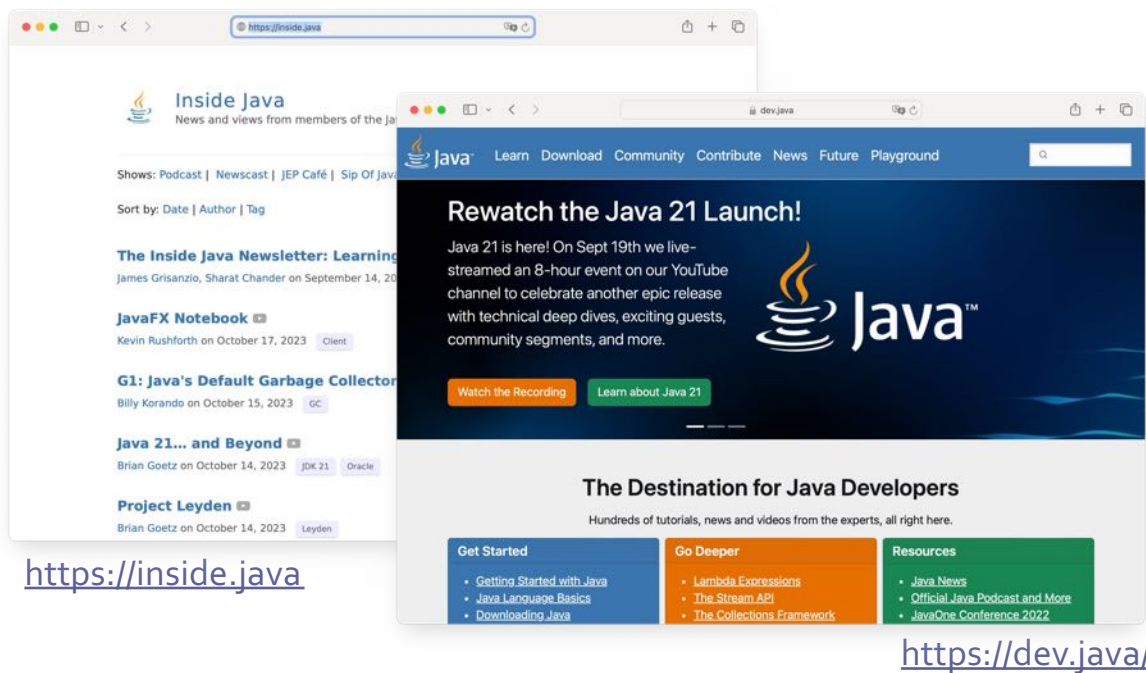


**Migration/Upgrade auf Java
11+ nicht
zu lange aufschieben.**

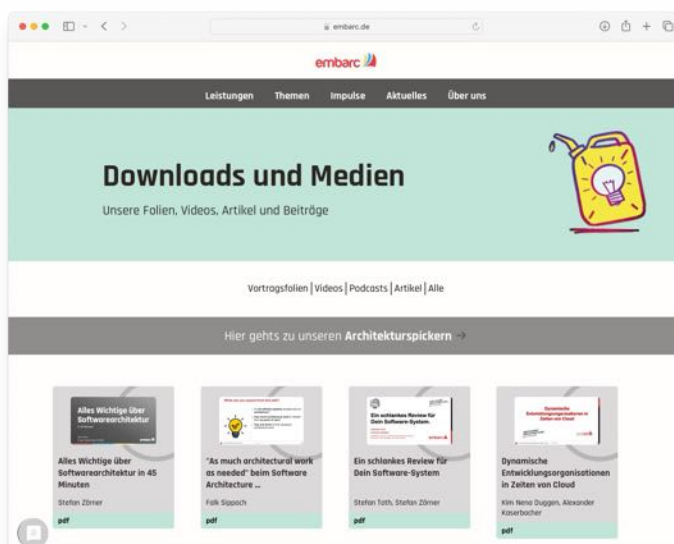
**Das nächste LTS Release ist
nicht fern ...** (alle 2 Jahre)



142



Folien als PDF zum Download



→ embarc.de/download/

Architektur-Spicker (1 -15)



„Mit unseren Architektur-Spickern beleuchten wir die konzeptionelle Seite der Softwareentwicklung.“

Architektur-Spicker #1 Der Architekturüberblick



PDF, 4 Seiten
Kostenloser Download.

→ embarc.de/architektur-spicker/

147

Vielen Dank.

Ich freue mich auf Eure Fragen!

-  falk.sippach@embarc.de
-  [linkedin.com/in/falk-sippach](https://www.linkedin.com/in/falk-sippach)
-  [@sippsack](https://twitter.com/sippsack)
-  [@sippsack@ijug.social](https://mastodon.social/@sippsack)



149



Falk Sippach

- Softwarearchitekt, Berater, Trainer bei embarc
- früher bei Orientation in Objects (OIO), Trivadis

Schwerpunkte

- Architekturberatung und -bewertung
- Cloud- und Java-Technologien

