Java wird 30

Die Neuerungen in Java 21 bis 25

Falk Sippach

entwickler.de 08.09.2025

embarc

0

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.

1

Falk Sippach

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

Schwerpunkte

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

embarc

2

Teil der Java Community

Nächster Termin am 23.10.

monatliche Opensource Code Camps

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

https://cyberland.ijug.eu/

3

Agenda

Was euch erwartet

01 Java – still a thing?

02 Die neuen Features

03 Was ist sonst passiert?

04 Fazit + Ausblick

4

01. Halbjährlicher Release-Prozess

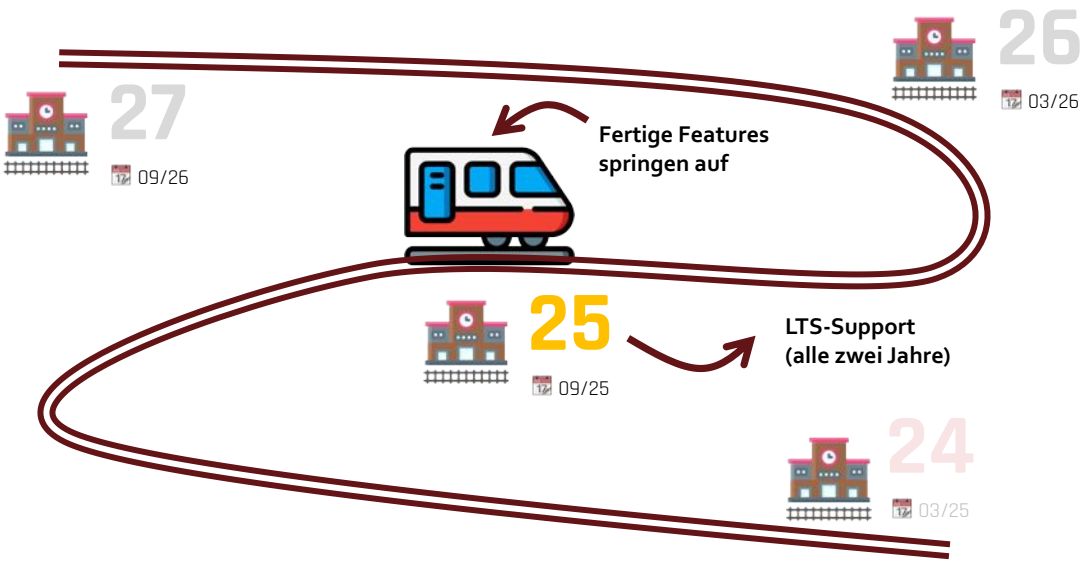
Seit über 5 Jahren wird halbjährlich eine neue Version des OpenJDK herausgebracht. Wie läuft der Prozess?

5

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

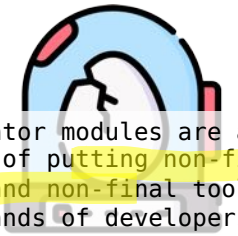
6

Halbjährlicher Release-Train



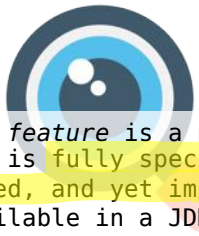
7

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

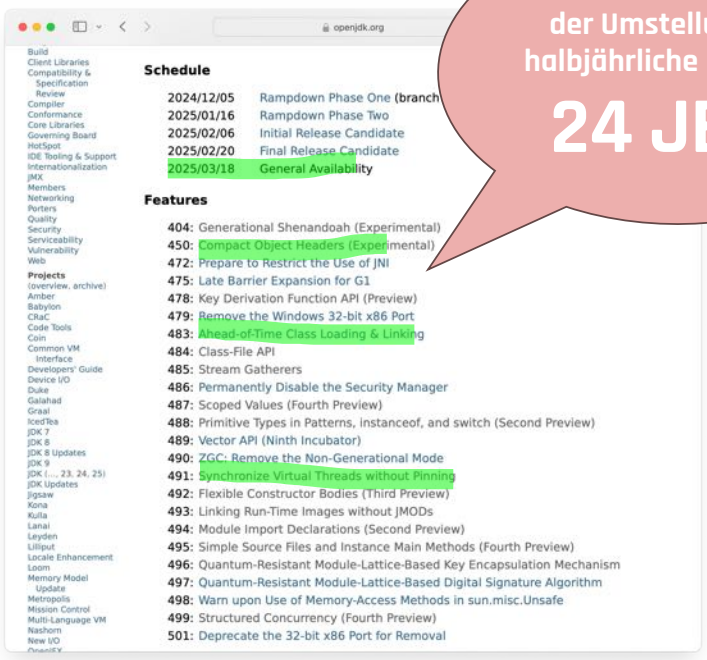


A *preview feature* is a new feature [..] that is fully specified, fully implemented, and yet impermanent. It is available in a JDK feature release to provoke developer feedback based on real world use; this may lead to it becoming permanent in a future Java SE Platform.

https://openjdk.org/jeps/12

9

Java 24

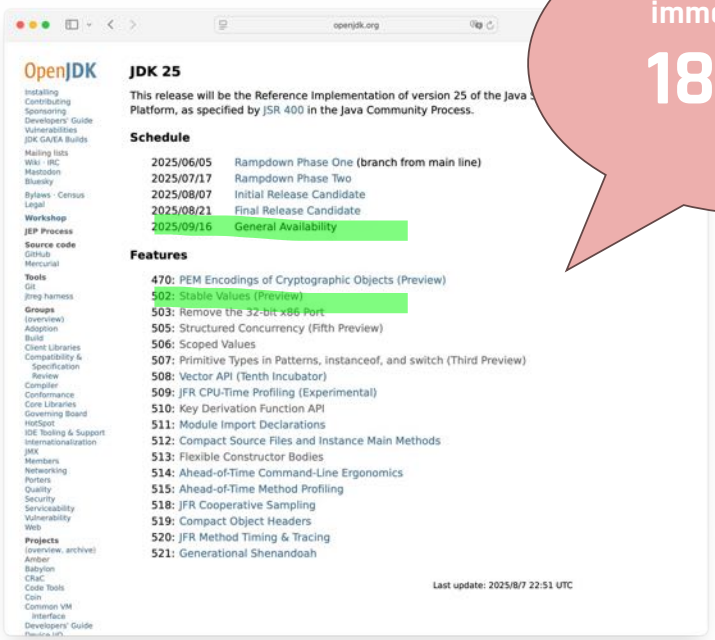


Neuer Rekord seit der Umstellung auf halbjährliche Releases:

24 JEPs

14

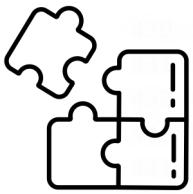
Java 25



immerhin wieder:

18 JEPs

15



Neue Features

Garbage Collectoren

Interna

API-Änderungen (JDK)

17

02.

Die neuen Features

Was ist überraschend, was ist besonders relevant?

18

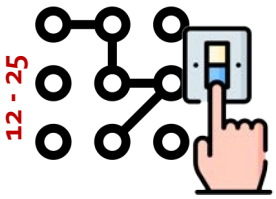
Neue Features

Garbage Collectoren

Interna

API-Änderungen (JDK)

19

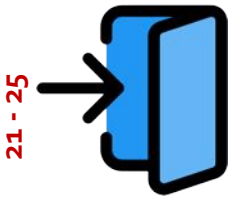


12 - 25

Pattern Matching

19 - 25

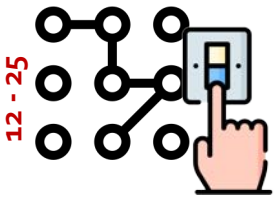
Rund um Virtual Threads



21 - 25

Paving the on-ramp

23



12 - 25

Pattern Matching

19 - 25

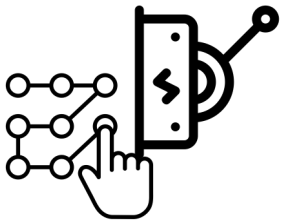
Rund um Virtual Threads

21 - 25

Paving the on-ramp

24

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

25

Pattern Matching

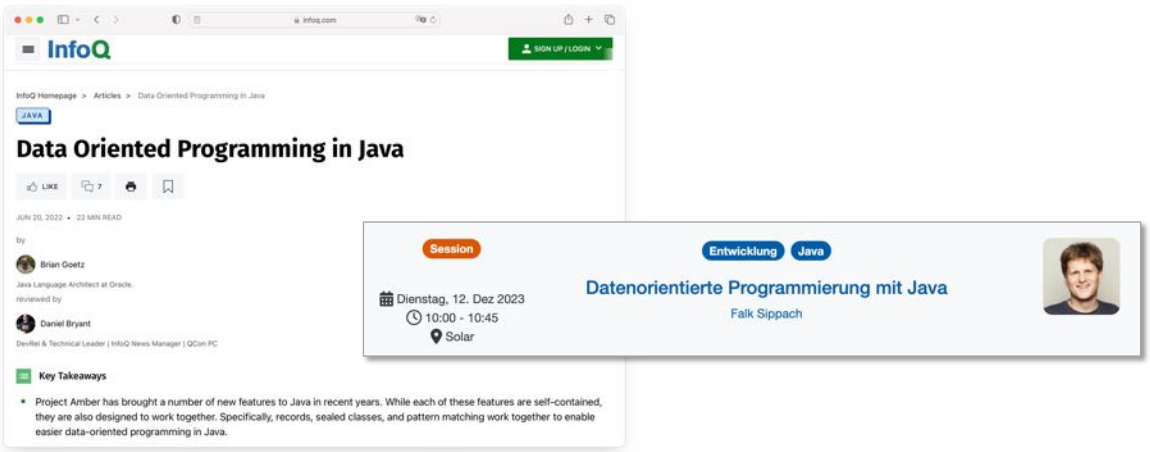
*„Pattern matching is a mechanism for **checking a value against a pattern**. A successful match can also **deconstruct a value into its constituent parts**.*

*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

26

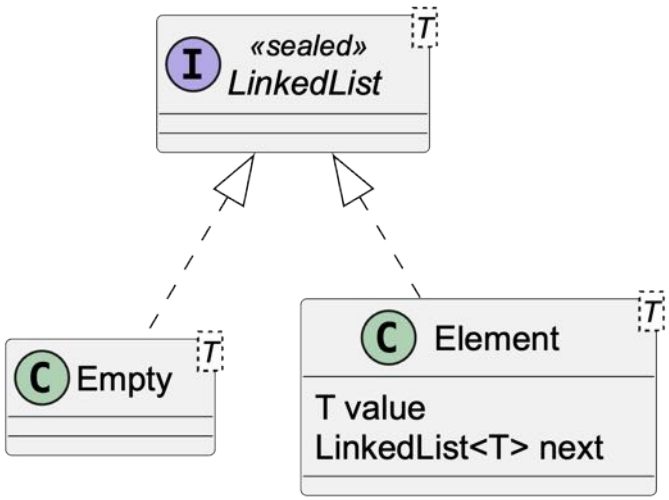
Neues Programmierparadigma?



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

27

Algebraische Datentypen

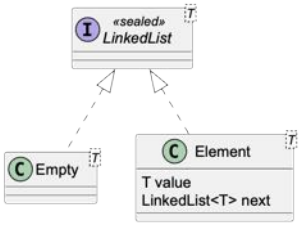


29

Sealed Class/Interface

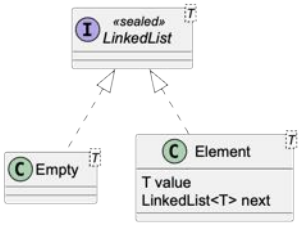
```
public sealed interface LinkedList<T>
  permits LinkedList.Element, LinkedList.Empty {

  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) {
    ..
  }
}
```

Record



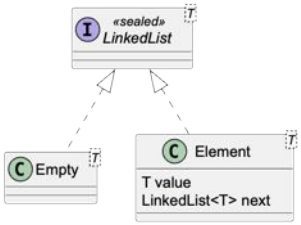
30


```
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;
}
```



31

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



32

switch-Expression

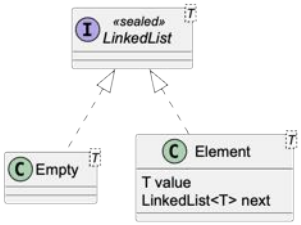
```
static <T> boolean contains(T value, LinkedList<T> list) {
  return switch (list) {
    case Empty empty -> false;
    case Element<T>(T v, var tail)
      when Objects.equals(v, value) -> true;
    case Element<T>(T v, var tail) -> contains(value, tail);
  };
}
```

Type Pattern
(Pattern Matching for instanceof)

Record Pattern

when-Clause (Guarded Pattern)

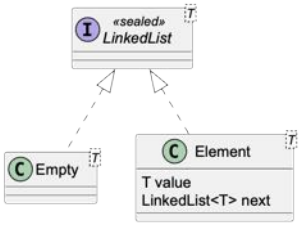
Kein Default notwendig (Exhaustiveness-Prüfung durch Compiler)!



33

```
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



36

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]
```

37

Primitive Type Patterns (1)

```
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 (2)

```
// dominierender vor dominierten Typ geht nicht (int vor byte)
value = 10;
switch (value) {
   // case int _ -> System.out.println(value + " instanceof int");
   case byte _ -> System.out.println(value + " instanceof byte");
   case int _ -> System.out.println(value + " instanceof int");
}
```

12 - 25

Pattern Matching

19 - 25

Rund um Virtual Threads

21 - 25

Paving the on-ramp

41

Project Loom

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

42

Die Geschichte von Threads in Java

1.1
1997
Green Threads

1.3
1998
Native Threads

5
2004
Thread Pools/**Executor**

7
2011
Fork-Join-Pools
Parallel Streams

8
2014 - 2016
Completable Future,
RxJava, Reactive
Streams, Reactor

9
2017 - 2018
Flow
Kotlin Coroutines

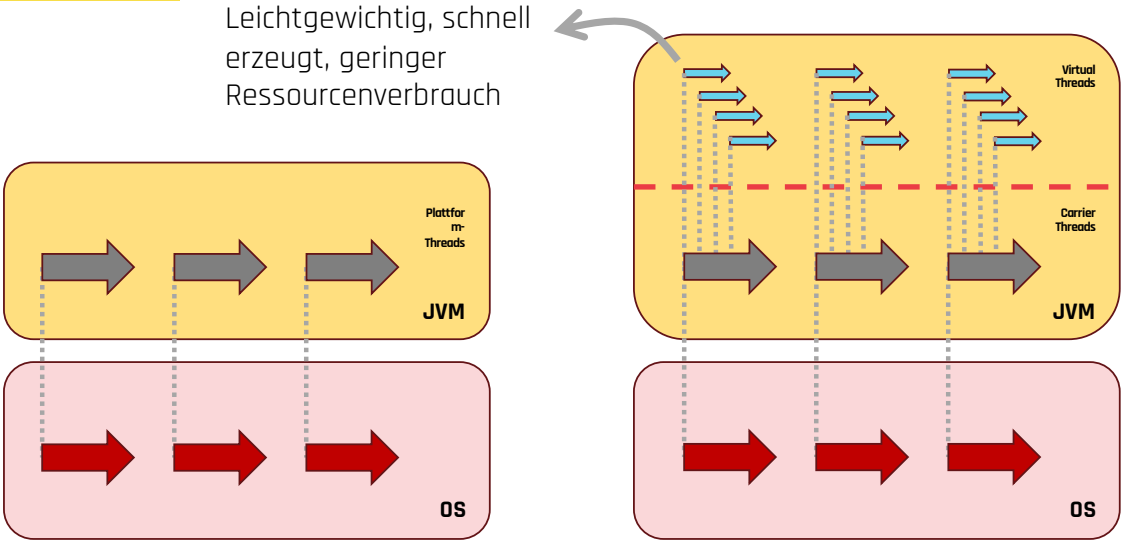
19
2022
Virtual Thread
(Preview)

21
2023
Virtual Threads Final

43

Plattform- vs. Virtual Threads

Leichtgewichtig, schnell
erzeugt, geringer
Ressourcenverbrauch



44

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());
});
```

Virtual Threads

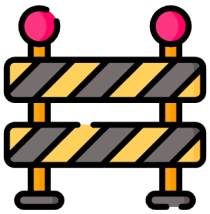
46

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

47

Pinning bei synchronize

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

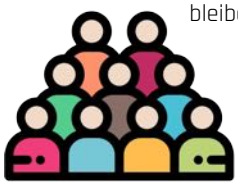
24

03/25

Für wen ist es gut?

viele/mehr parallele Requests

aber **externe Ressourcen** bleiben **Bottleneck**



nicht für die Verarbeitung großer Datenmengen bzw. **langlaufende Berechnungen**

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

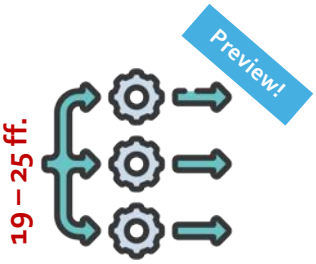
Virtual Threads unter der Haube

Wir alle, die Webanwendungen bauen und davon **indirekt profitieren**.

Nebenläufige Programmierung

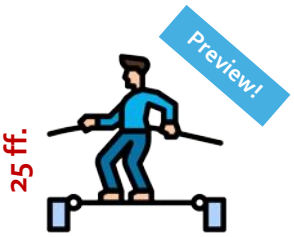
Implementierung von **sehr vielen** und/oder **blockierenden Aufgaben**

Weiteres im Umfeld von Virtual Threads



Structured Concurrency

Scoped Values

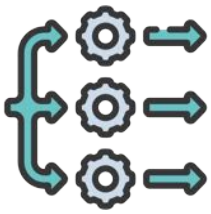


Stable Values

Funktioniert auch mit **Platform Threads!**

51

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**

52

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
- …

57

Structured Concurrency

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

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

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());
  }
}

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

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

basiert auf Virtual Threads

59

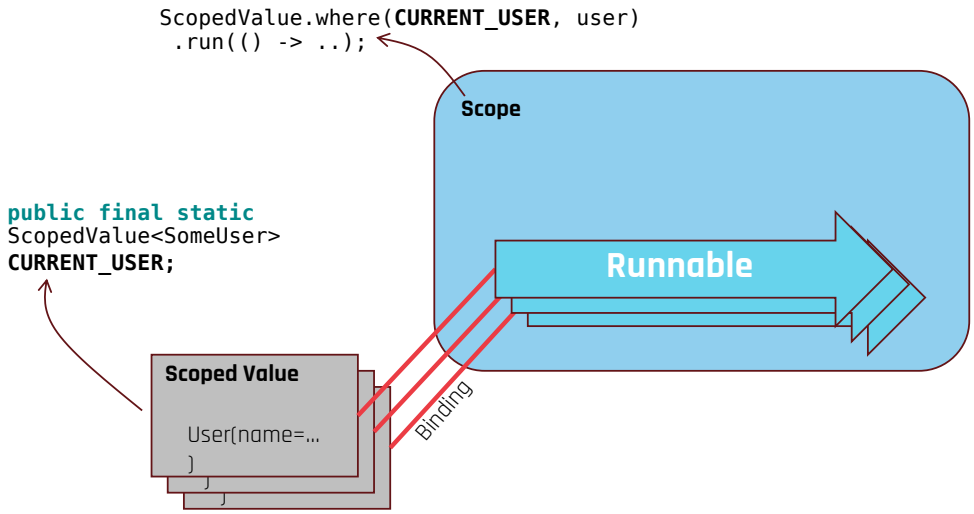
Structured Concurrency - Strategien

```
// 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")))
```

basiert auf
Virtual Threads

60

Scoped Values - Alternative zu ThreadLocal



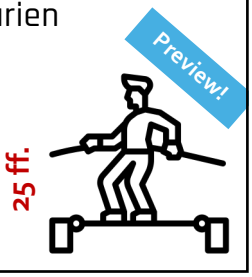
64

Scoped Values

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

public void someControllerAction(HttpRequest 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

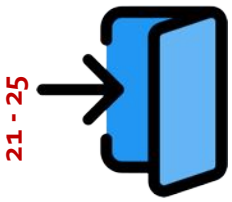


Beispiel Stable Value

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

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


Pattern Matching

Rund um Virtual Threads



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
```


71

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

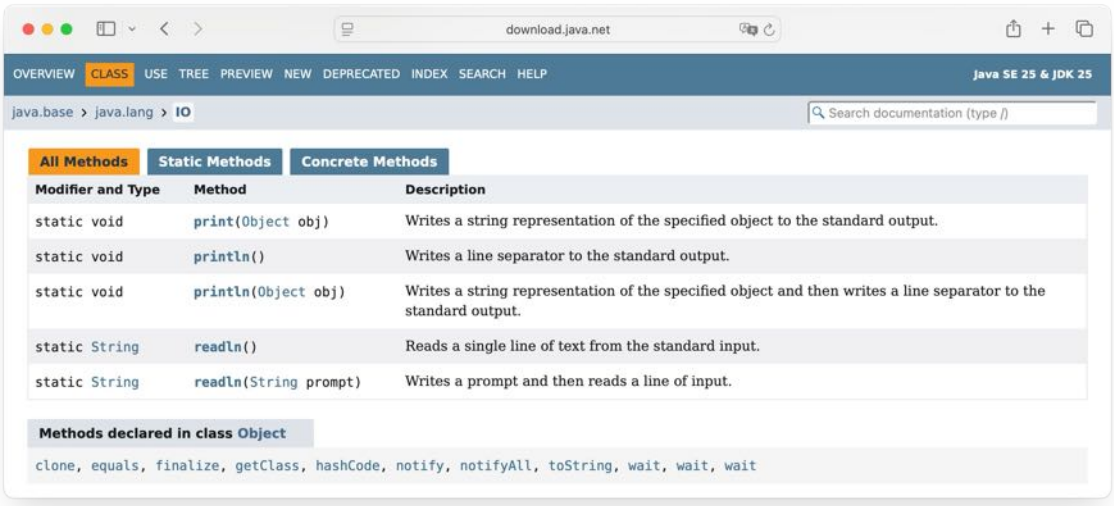
72

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



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());
  }
```


Pattern Matching

Rund um Virtual Threads

Was ist noch hinzugekommen?

Paving the on-ramp

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;
}
```


81

Markdown in Javadoc

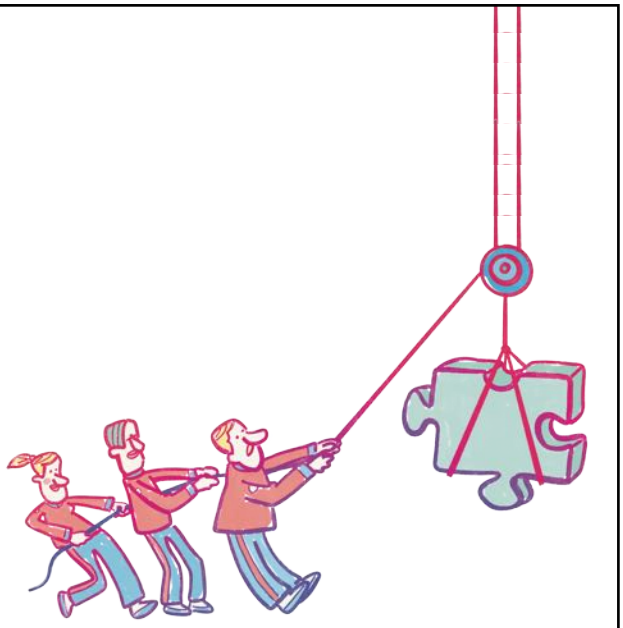
```
/// Returns the ... of `int` and `java.lang.String` arguments.
///
/// Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam
///  nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam
///  erat, sed diam voluptua.
///
/// At vero eos et accusam et justo duo dolores et ea rebum.
/// - 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;
}
```


82

03.

Was ist sonst passiert?

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



83

Neue Features

Garbage Collectoren

Interna

API-Änderungen (JDK)

84

Neue Features

Garbage Collectoren

Interna

API-Änderungen (JDK)

85

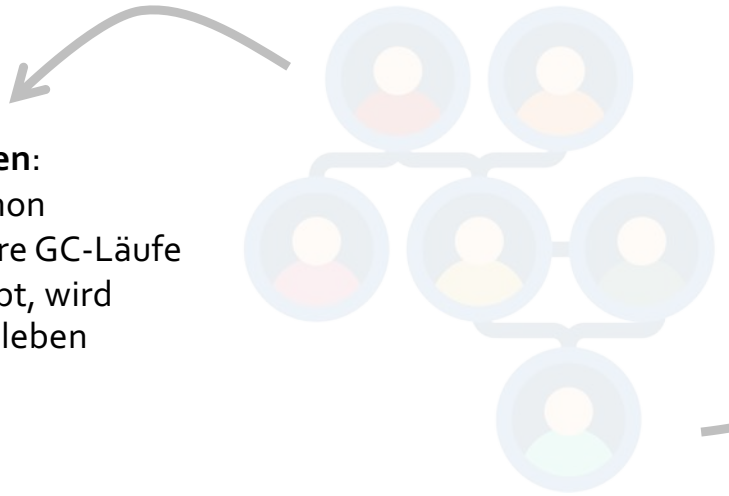
Neue Generation von Low-Latency
Garbage Collectoren: ZGC und Shennadoah

moderne Architekturen: Multi-Core + TB RAM

Ziel: kurze GC-Pausen im ms-Bereich

86

JEP 439: Generational ZGC



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

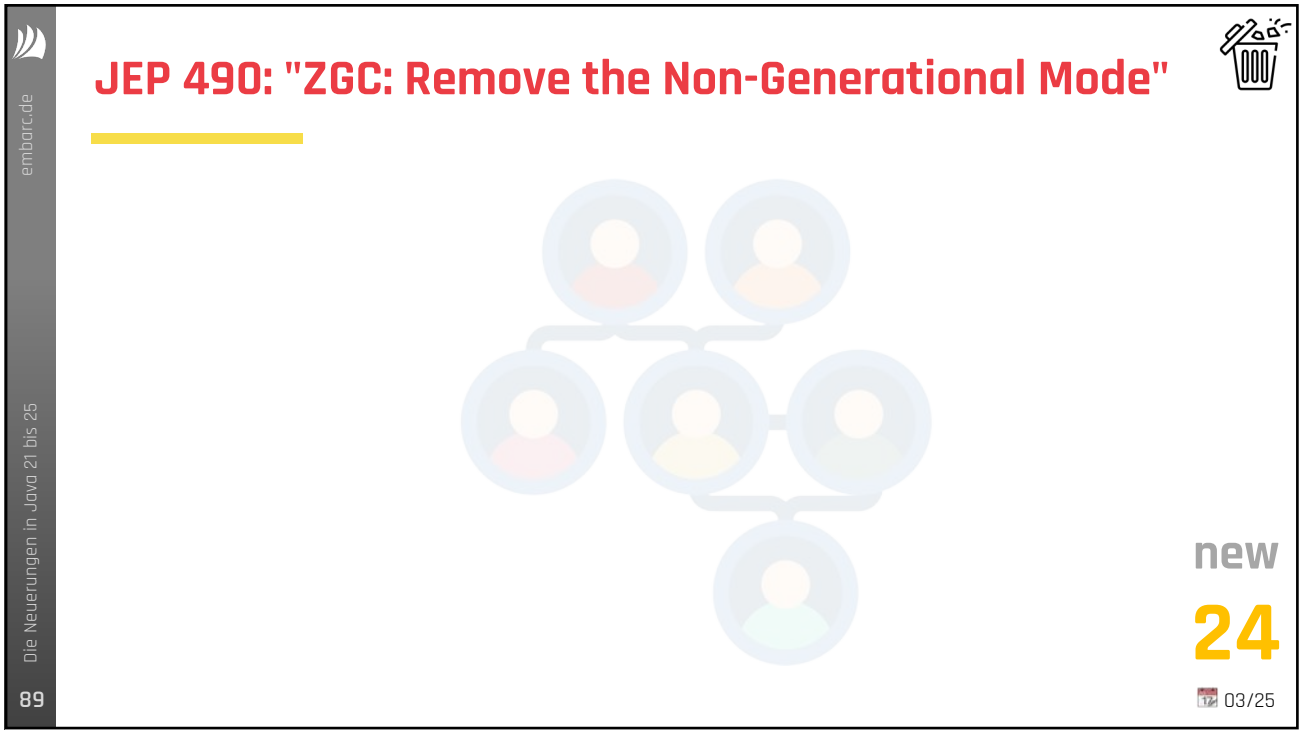
Young-Gen:
wird im GC-Lauf prozessiert

87

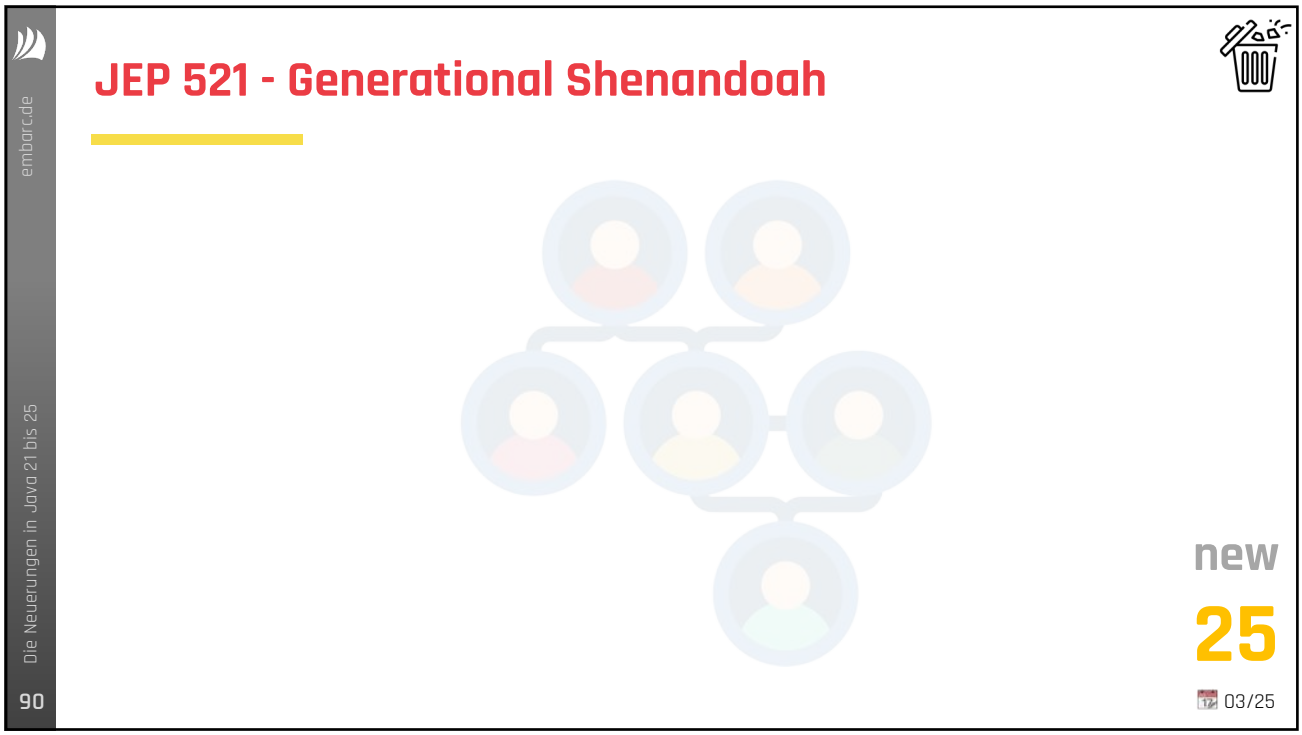
JEP 474: ZGC: Generational Mode by Default

G1 ist weiterhin der
Default Garbage Collector

88


embarc.de
Die Neuerungen in Java 21 bis 25
89
JEP 490: "ZGC: Remove the Non-Generational Mode"
new
24
03/25

89


embarc.de
Die Neuerungen in Java 21 bis 25
90
JEP 521 - Generational Shenandoah
new
25
03/25

90

Neue Features

Garbage Collectoren

Interna

API-Änderungen (JDK)

91

Finalisiert im **OpenJDK 22!**

14 – 22

Foreign Linker + Foreign-Memory Access API

Wartet auf die Reformen des Typsystems (**Value Classes**)!

Inkubator!

14 – 22 ff.

Vector API

92

Class File API (Ersatz für ASM)

finalisiert in **Java 24**

Einsatz von **Pattern Matching** für Analyse und Modifizierung!

```
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 ...
  }
}
```


Class File API

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

```
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_();
  }
);
```

Methode **foobar** erzeugen!

Class File API

In bestehender Klasse alle **debug**-Methoden **löschen**!

```
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);
      }
    }
  }
);
```


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

97

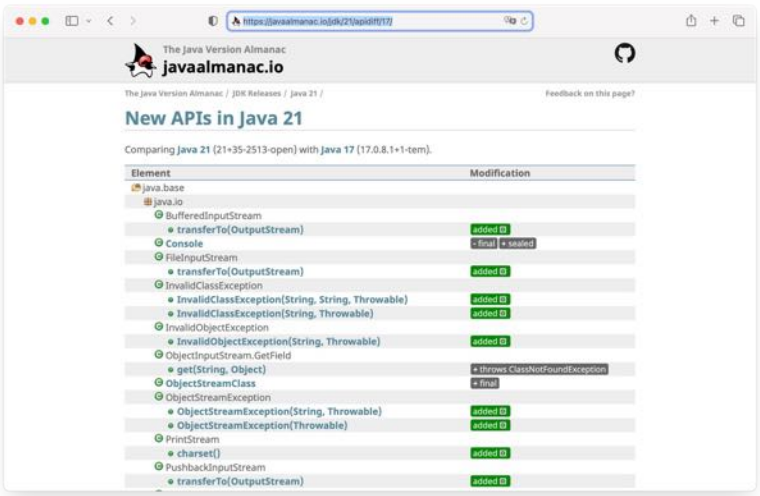
Neue Features

Garbage Collectoren

Interna

API-Änderungen (JDK)

98



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

99

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**

104

Analogie Collector API

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

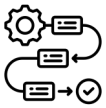
Beispiel: windowFixed

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**!

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!

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

Eigene Gatherer

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

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

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

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

Optional: Zustand über alle Stream-Elemente speichern.

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

Optional: Zusammenbringen einer parallelen Verarbeitung.

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

109

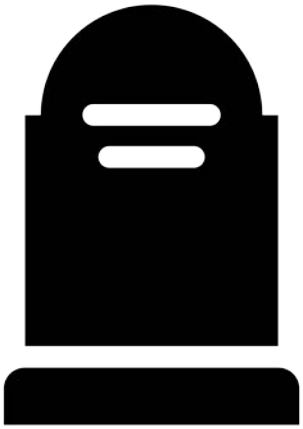
JEP 496/497: Quantum-Resistant Module-Lattice-Based …

- … 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 Algorithmus

new
24
03/25

116

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

119

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

120

04.

Fazit und Ausblick

Zusammenfassung und Zukunft

121

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
```

122

122

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

123

Halbjährlicher Release-Train

Neues Release-Modell

+

Fertige Features
springen auf

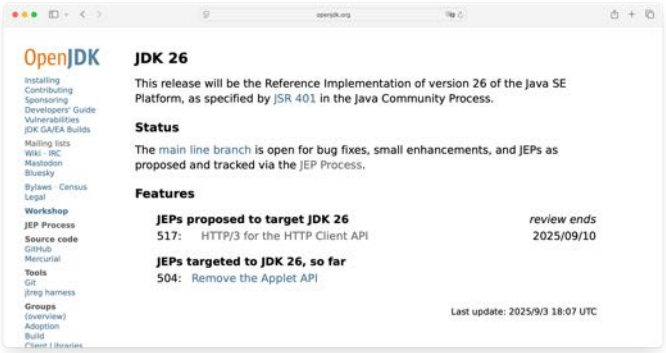
Feedback-Mechanismen

LTS-Support
(alle zwei Jahre)

funktionieren!

124

Java 26



pre
26
03/26

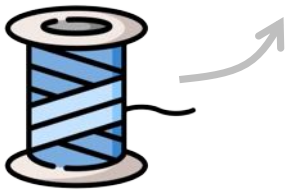
126

Pattern Matching + kleine Syntax-Verbesserungen

Amber

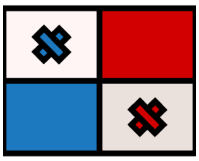
Was kommt noch?

Virtual Threads



Loom

Vector API + Foreign Function & Memory API



Panama

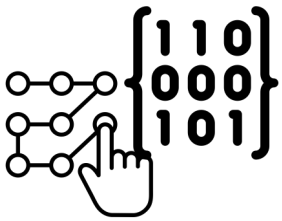
Valhalla

Value Types

127

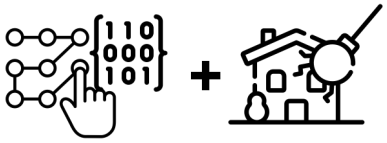
Array Patterns

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



Kombination: Array & Records Patterns

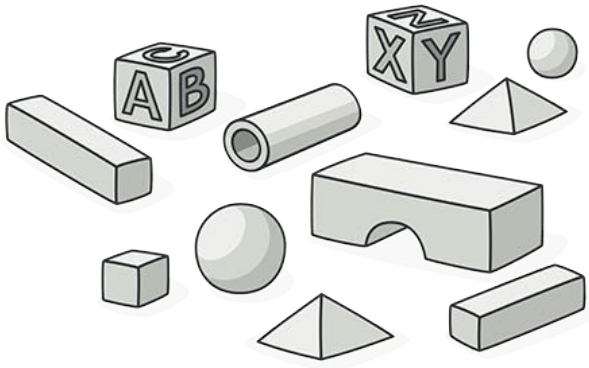
```
static void printSumOfFirstTwoXCoords(Object o) {
  if (o instanceof Point[] {
              Point(var x1, var y1),
              Point(var x2, var y2),
              ... }) {
    System.out.println(x1 + x2);
  }
}
```



Record Patterns in foreach-Schleifen

130

Primitive Obsession



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

131

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**

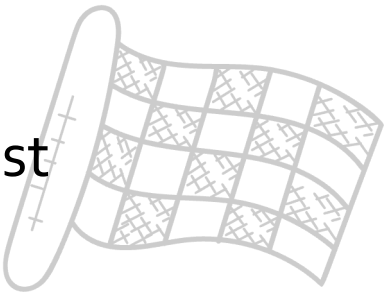
Nicht alle benutzer-definiertren Typen brauchen das

Value Types

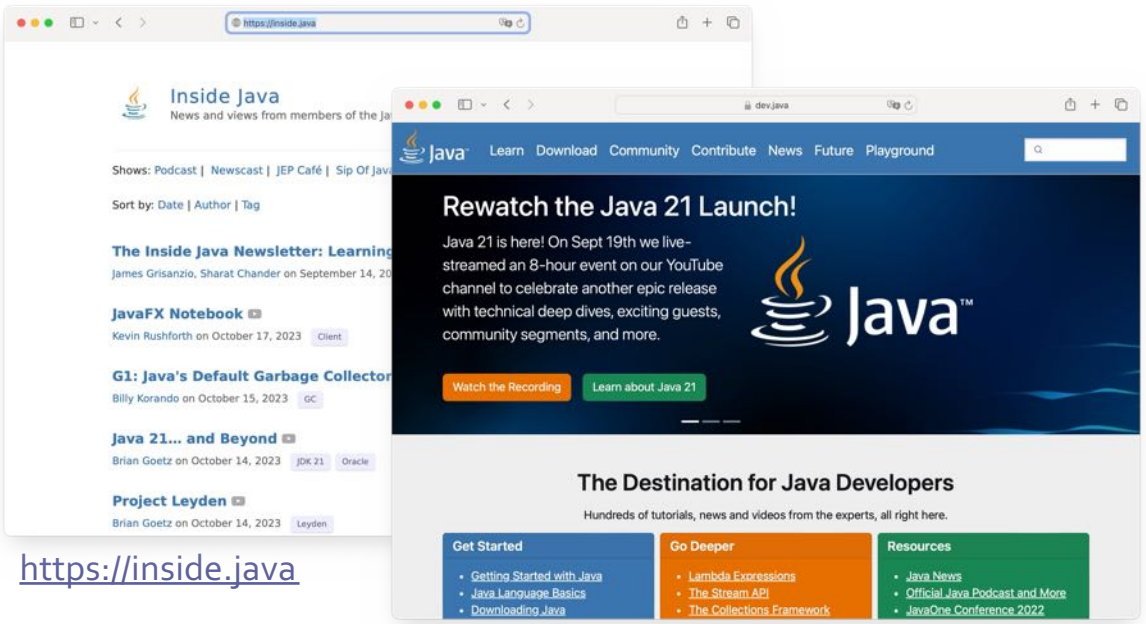
```
value class Point {
  private int x;
  private int y;

  public implicit Point();

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

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

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

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



134

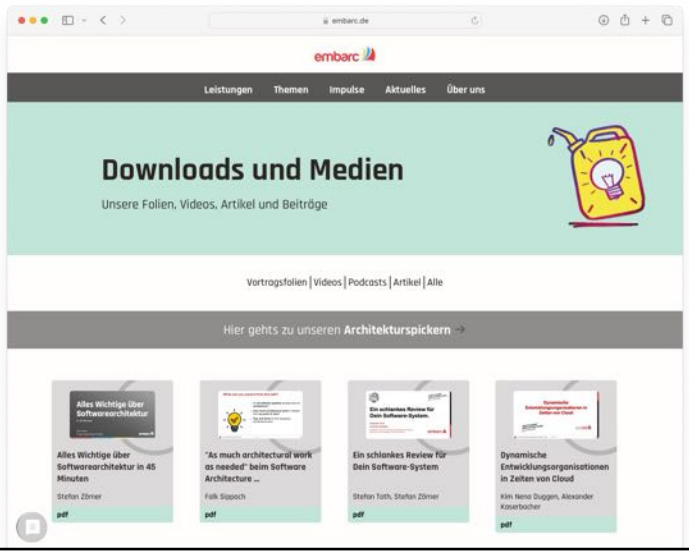


https://inside.java

https://dev.java/

135

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/

Vielen Dank.

Ich freue mich auf Eure Fragen!

falk.sippach@embarc.de
linkedin.com/in/falk-sippach
@sippsack
@sippsack@ijug.social

embarc

141

Falk Sippach

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

Schwerpunkte

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

embarc

142