embarc
Software Consulting GmbH

Die nächste Version
mit **LTS-Support**

Java 21

~~Die nächste LTS-Version~~

FALK SIPPACH // EMBARC

JUG SAXONY DAY

1

Java 21 - Die nächste Version mit LTS-Support

Alle halben Jahre erscheinen neue Major-Releases, alle zwei Jahre als LTS mit längerer Unterstützung (Long Term Support). Viele arbeiten aber noch mit der früheren LTS-Versionen 8 oder sind gerade erst auf 11 umgestiegen. Wir wollen uns daher anschauen, was sich seitdem im Java-Universum getan hat. Das sind so spannende Themen wie Pattern Matching und Virtual Threads. Aber es sind noch viele weitere Features in Arbeit oder der Planung, wie die Vector API, die Foreign Function & Memory API, die String Interpolation oder die Primitive bzw. Value Types.

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.

2

Falk Sippach

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

Schwerpunkte:

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

fs@embarc.de
@sippsack
➔ xing.to/fsi

embarc Software Consulting GmbH

3

Agenda

1 Java – still a thing?

2 Ist Java 21 ein LTS-Release?

3 Viele spannende neue Features

4 Fazit

7

Agenda

1 **Java – still a thing?**

2 Ist Java 21 ein LTS-Release?

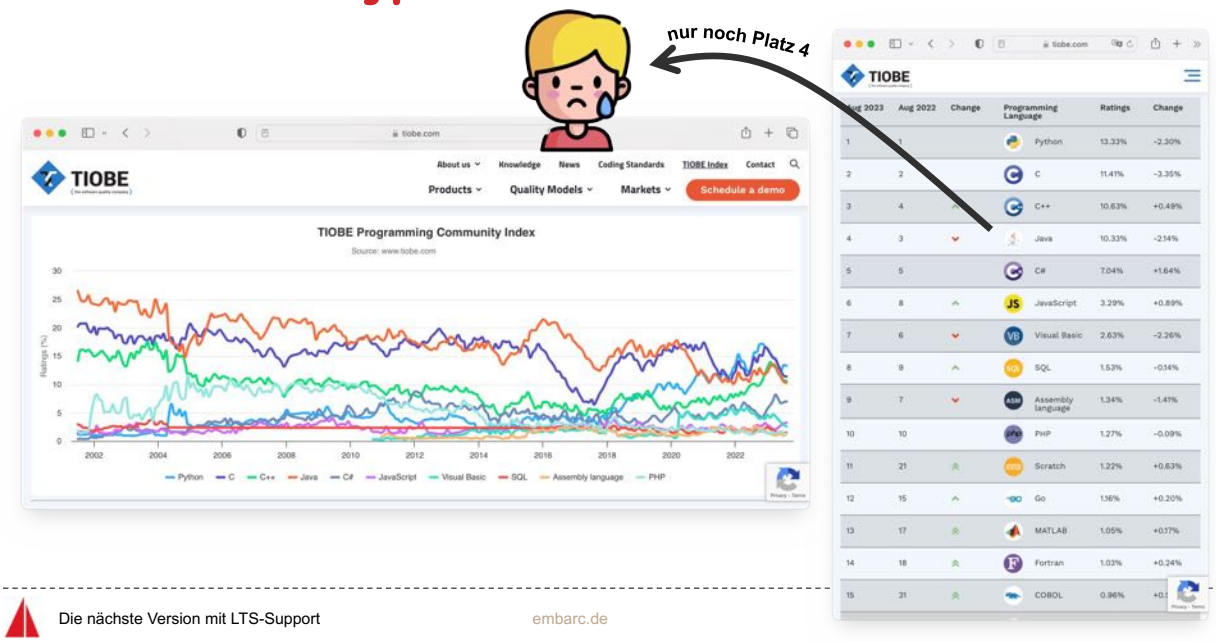
3 Viele spannende neue Features

4 Fazit

1

8

Is Java still hyped or still relevant at all?



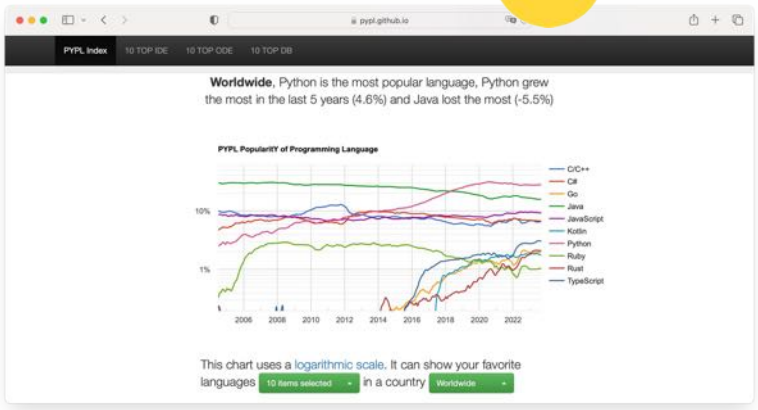
9

Traue keiner Statistik, die Du nicht selbst …

10

PYPL: PopularitY of Programming Language

immerhin schon Platz 2

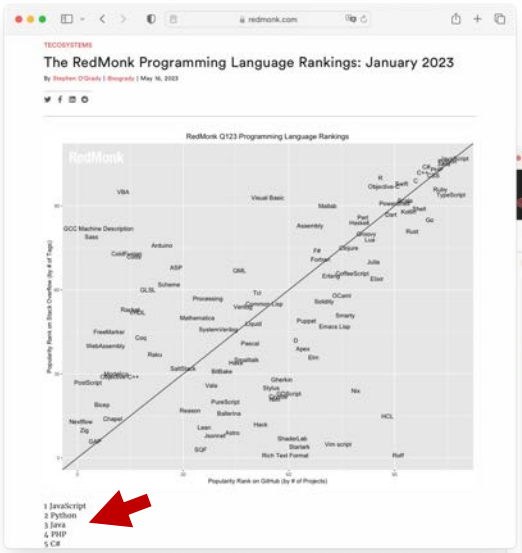


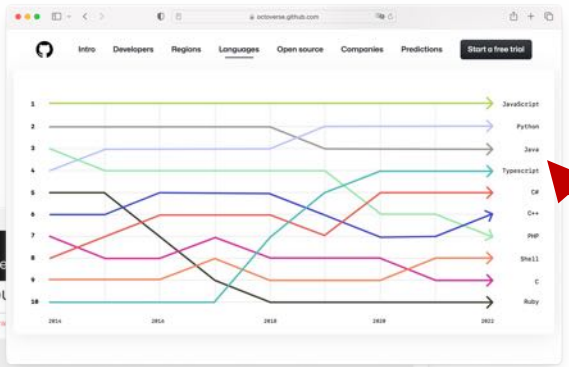
Worldwide, Sept 2023 :

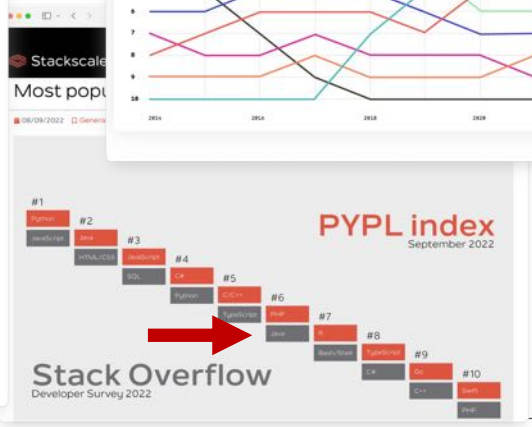
Rank	Change	Language	Share	1-year trend
1		Python	27.99 %	+0.1 %
2		Java	15.9 %	-1.1 %
3		JavaScript	9.36 %	-0.1 %
4		C#	6.67 %	-0.4 %
5		C/C++	6.54 %	+0.3 %
6		PHP	4.91 %	-0.4 %
7		R	4.4 %	+0.2 %
8		TypeScript	3.04 %	+0.2 %
9	↑↑	Swift	2.64 %	+0.6 %
10		Objective-C	2.15 %	+0.1 %
11	↑↑	Rust	2.12 %	+0.5 %
12	↓↓↓	Go	2.0 %	-0.1 %
13	↓	Kotlin	1.78 %	-0.0 %

11

Viele Statistiken später

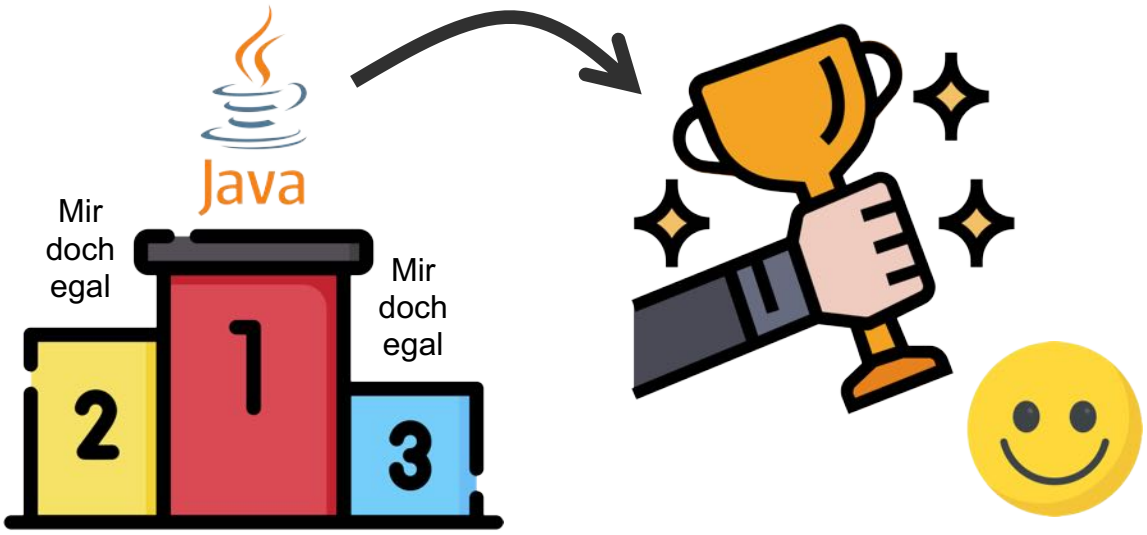






12

FPSI: Falks Programmiersprachen Index

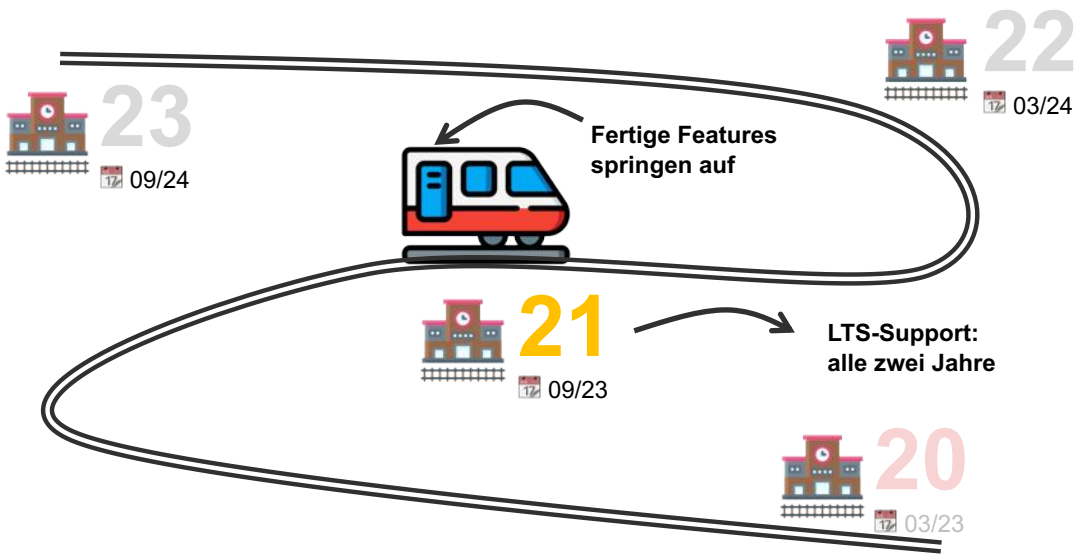


13

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

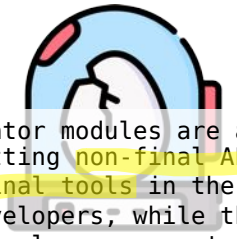
14

Halbjährlicher Release-Train



15

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

Agenda

1 Java – still a thing?

2 **Ist Java 21 ein LTS-Release?**

3 Viele spannende neue Features

4 Fazit

2

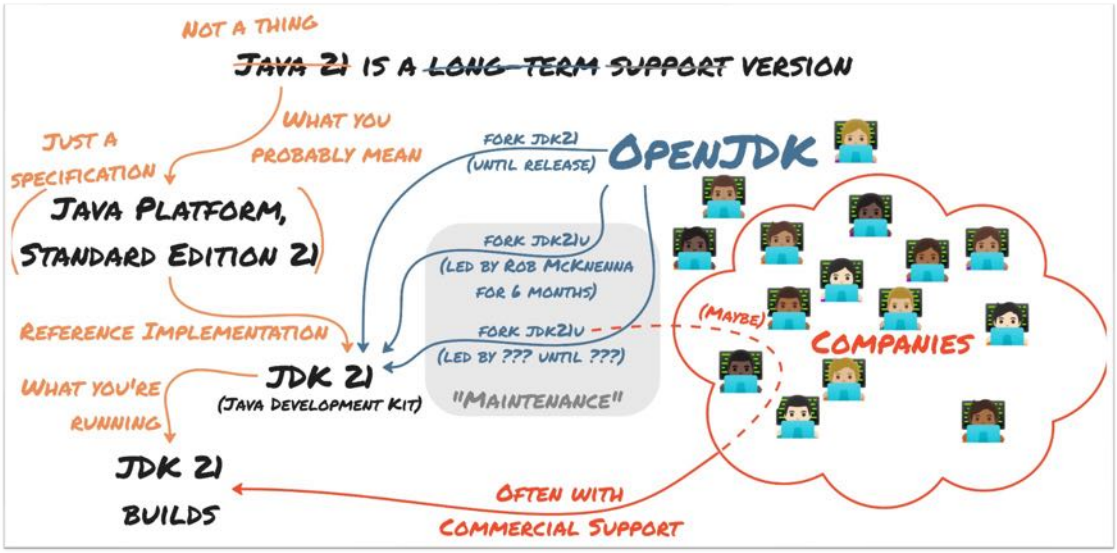
Ist 21 Java überhaupt ein LTS-Release?

18

Ist Java 21 überhaupt ein LTS-Release?

19

Diverse JDK 21 Build erhalten LTS …



https://twitter.com/nipafx/status/1676908785313492992?t=9K-kilkLOSbS6RfT9OYKzw&s=03

20

Zoo von JDKs

OpenJDK

ADOPTIUM

Amazon Corretto 8
Now Generally Available
aws

AZUL SYSTEMS

Sap Machine

OpenJDK

IBM

OpenJ9

Alibaba Dragonwell

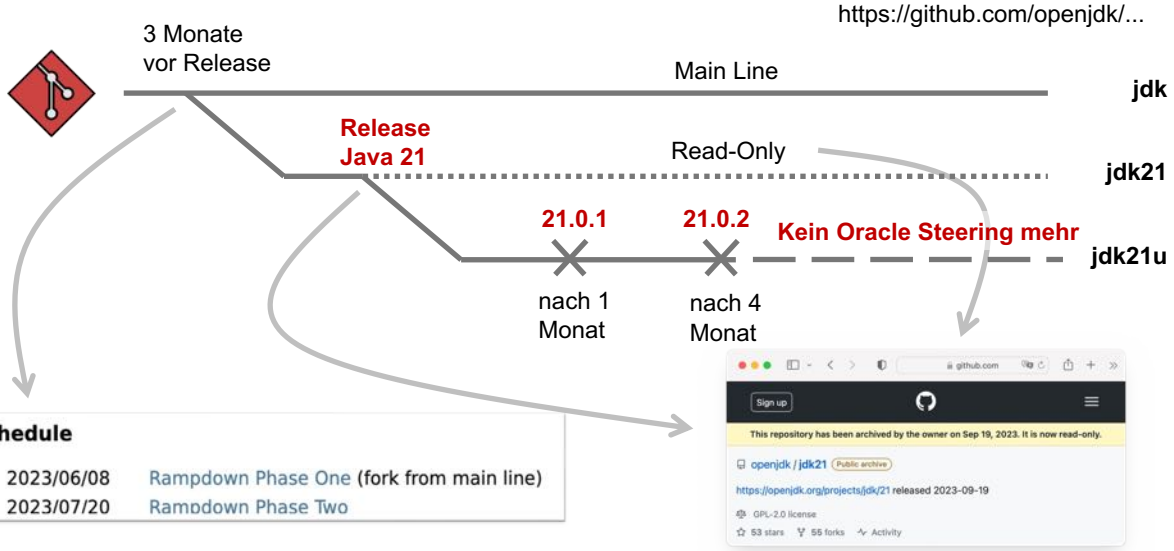
LIBERICA JDK

Java
ORACLE

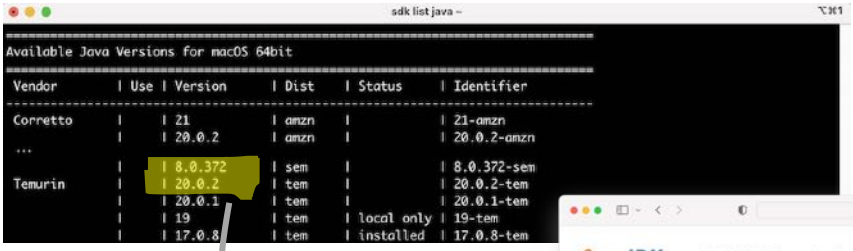
21

JEP 3: JDK Release Process

https://github.com/openjdk/...



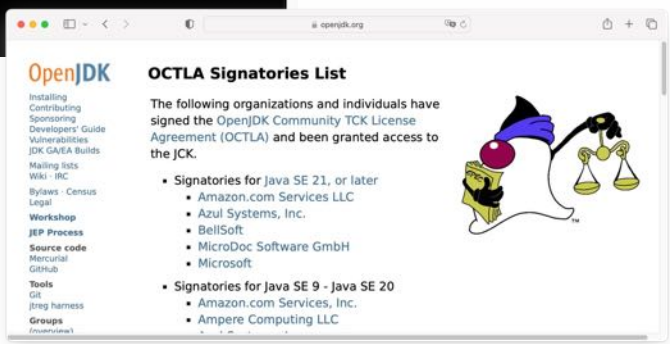
Apropos LTS Release ...

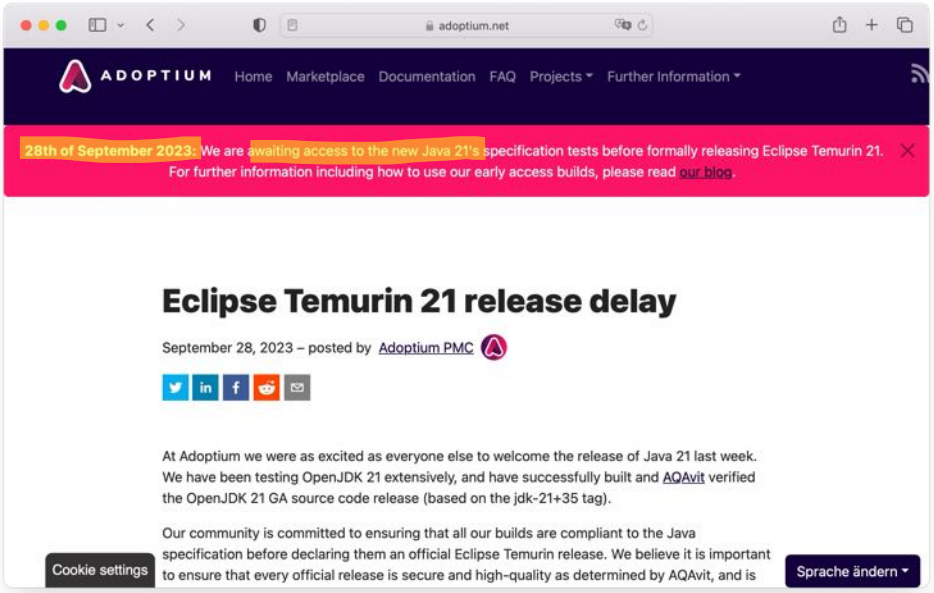


Screenshots vom 29.09.2023

TEMURIN by ADOPTIUM

Warum gibt es 10 Tage nach dem Java 21 Release noch keine Temurin Variante?





24

Agenda

1 Java – still a thing?

2 Ist Java 21 ein LTS-Release?

3 **Viele spannende neue Features**

4 Fazit

3

25

Java 21 is …

26

LTS

Java **17**	Java **18**	Java **19**	Java **20**	…
14 JEPs	**9** JEPs	**7** JEPs	**7** JEPs	
Pattern Matching	*Pattern Matching*	*Pattern Matching*	*Pattern Matching*	
Sealed Classes	*Foreign Function & Memory API*	*Foreign Function & Memory API*	*Foreign Function & Memory API*	
Records	*Vector API*	*Vector API*	*Vector API*	
Foreign Function & Memory API	**Simple Web Server**	*Virtual Threads*	*Virtual Threads*	
Vector API	**JavaDoc Code Snippets**	*Structured Concurrency*	*Structured Concurrency*	
			Scoped Values	

Legende: **final** – *Preview/Inkubator*

27

Pattern Matching

Unnamed Patterns

Foreign Function & Memory API

Vector API

Virtual Threads / Structured Concurrency /

Scoped Values

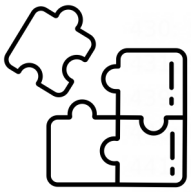
String Templates

Sequenced Collections

Unnamed Classes and Instance Main Methods

Java **21**: **15** JEPs

28



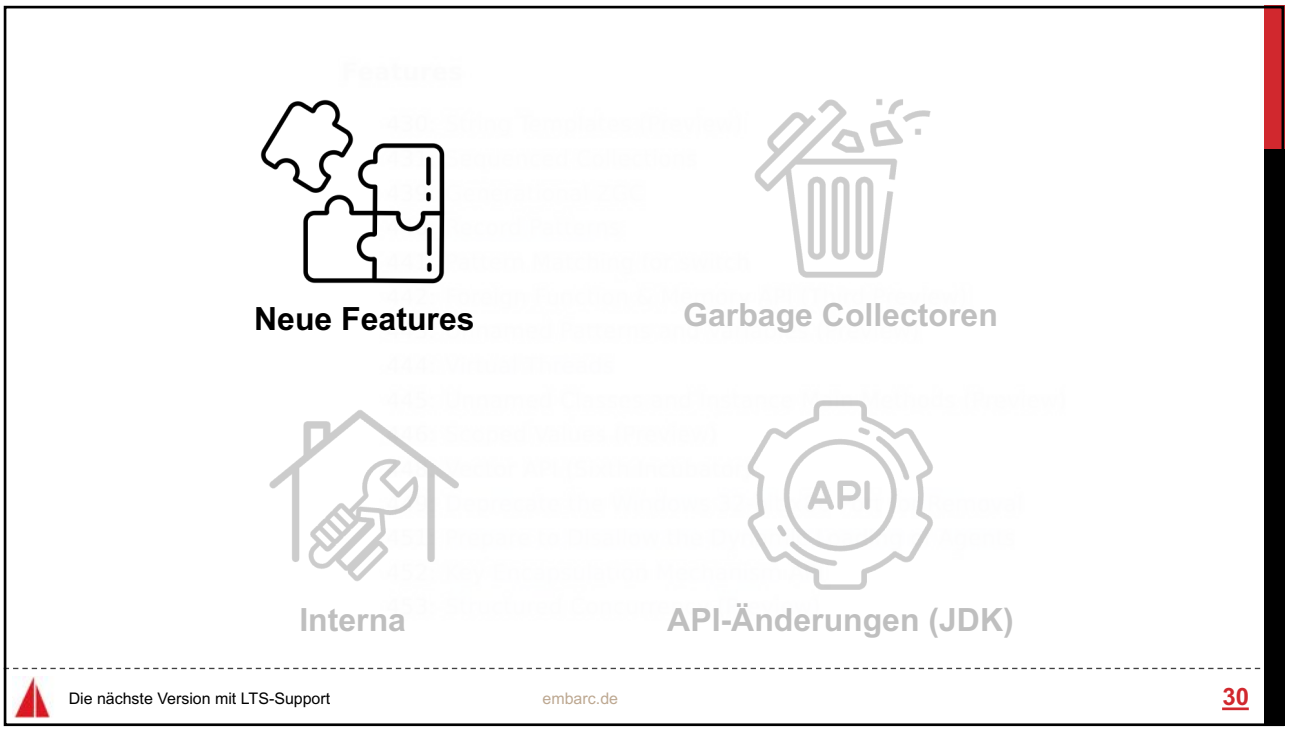
Neue Features

Garbage Collectoren

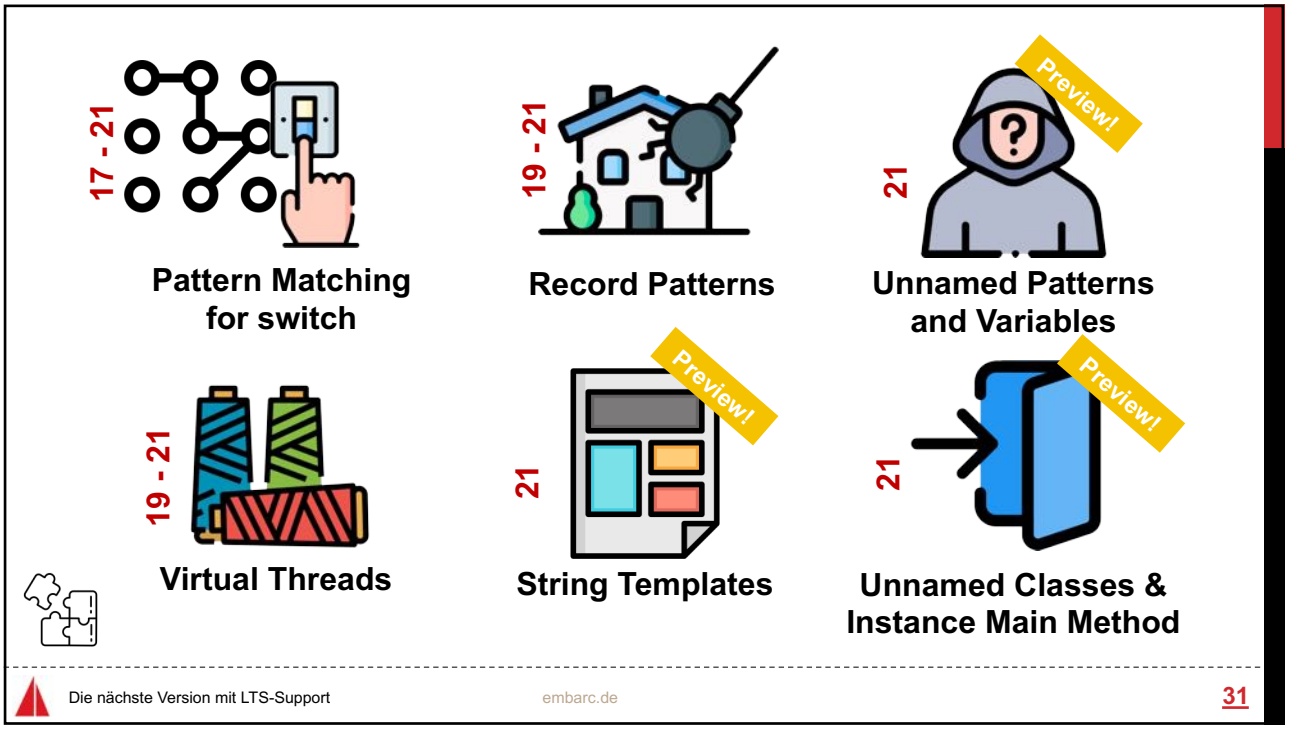
Interna

API-Änderungen (JDK)

29


Neue Features
Garbage Collectoren
Interna
API
API-Änderungen (JDK)
Die nächste Version mit LTS-Support
embarc.de
30

30


17 - 21
Pattern Matching for switch
19 - 21
Record Patterns
Preview!
21
Unnamed Patterns and Variables
19 - 21
Virtual Threads
Preview!
21
String Templates
Preview!
21
Unnamed Classes & Instance Main Method
Die nächste Version mit LTS-Support
embarc.de
31

31

Unnamed Classes & Instance Main Method

```
void main() {
   System.out.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 --source 21 --enable-preview Main.java
```


32

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!

33

Launch-Protokoll

Gewinner!

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

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

34

String Templates (String Interpolation)

```
String ersetzung = 'Ersetzungen'
String s = "Ein GString kann ${ersetzung} enthalten!"

println "Hallo ${user.name}!"
```

Triggerwarnung: Diese Folie ist Original von 2009!

35

Strings in Java

- String-Verkettung mit "+", StringBuffer/-Builder, String.format(), …
- ~~JEP 326: Raw String Literals (Preview)~~ in Java 12 (keine Sonderzeichen escapen)
- Text-Blocks ab Java 13 (mehrzeilige Strings)
- 430: **String Templates** (Preview) ab Java 21

36

```
User user = new User("dieter@develop.de",
    new Role("admin"));

String info = STR."User '\{user.email()}' hat die
Rolle '\{user.role().name()}'";

// User 'dieter@develop.de' hat die Rolle 'admin'
System.out.println(info);
```

37

Warum \{variable}?

Pre Java 21

```
public static void main(String[] args) {
   String s = "\{args}";
}
```

Kompiler-Fehler:
java: illegal escape character

Abwärtskompatibilität!

38

```
String title = "My Web Page";
String text = "Hello, world";
String html = STR."""
      <html>
        <head>
          <title> \{
            // Comment
            "My company: " + title
          }
          </title>
        </head>
        <body>
          <p>\{text}</p>
        </body>
      </html> """;
```


39

FormatProcessor

```
double zahl = 2.0;

String formatted = FMT."Zahl: %7.2f\{zahl}";

System.out.println(formatted); // Zahl:    2.00
```

40

RAW Processor

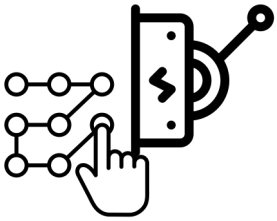
```
var template = RAW."Today is day
\{LocalDate.now().getDayOfYear()} of year
\{LocalDate.now().getYear()}.";

System.out.println(template.fragments());
System.out.println(template.values());
String result = template.process(STR);
System.out.println(result); // Today is day …
```

41

Custom Processor

```
StringTemplate.Processor<ResultSet, SQLException> SQL = template -> {
   String sql = template.fragments().stream().collect(Collectors.joining(" ? "));
   PreparedStatement preparedStatement = connection.prepareStatement(sql);
   ...
   return preparedStatement.executeQuery();
};

var result = SQL."""
        select id, email, enabled from Users u
        where u.email = \{user.email()}
        and u.enabled = \{user.enabled()};";
result.next();
System.out.println(STR."Found: \{result.getString("email")}");
```


42



Pattern Matching for switch

Record Patterns (Deconstruction)

Unnamed Patterns and Variables

43

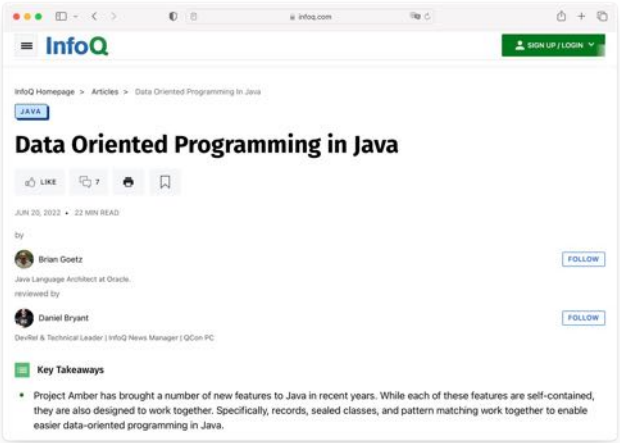
Pattern Matching

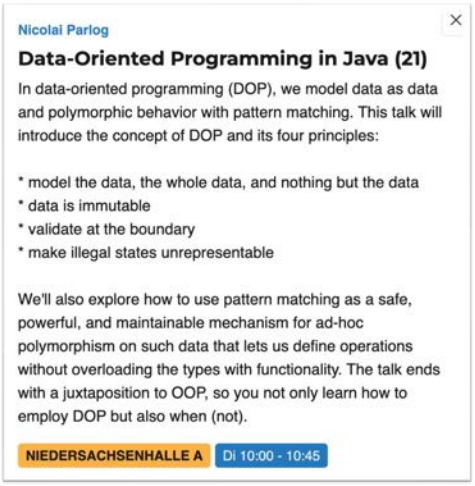
*„Pattern matching is a mechanism for **checking a value against a pattern**. A successful match can also **deconstruct a value into its constituent parts**.*

*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

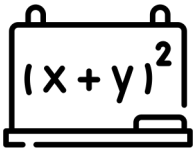
Neues Programmierparadigma?





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

https://slides.nipafx.dev/patterns/#/



Algebraische Datentypen

Sealed Classes

Records

Σ

Summentypen

Produkttypen

es gibt noch mehr: Quotienten-, Aufzählungstypen, …

46

Liste als algebraischer Datentyp

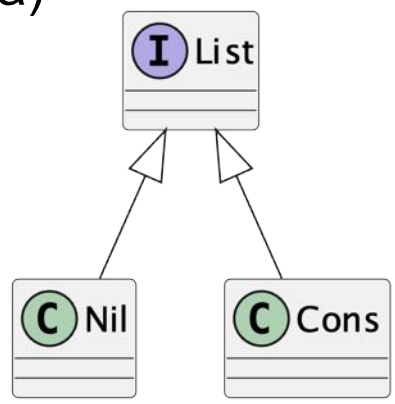
= Construct

```
data List a = Nil | Cons a (List a)
```

[ ]

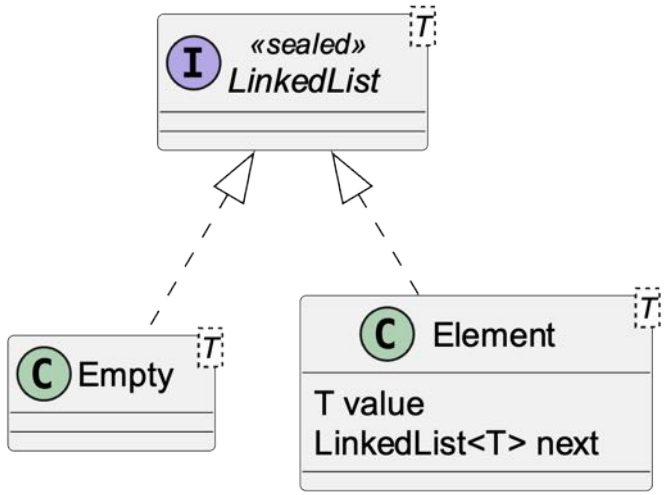
:: oder :

```
Cons 1 (Cons 2 (Cons 3 Nil))
1:2:3:[]
[1,2,3]
```



47

Algebraische Datentypen

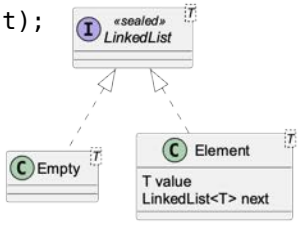


48

Sealed Class/Interface

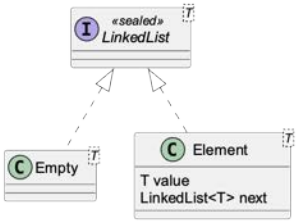
Record

```
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) {
    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;
  }
}
```



49

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



50

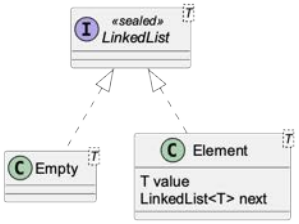
switch-Expression

```
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

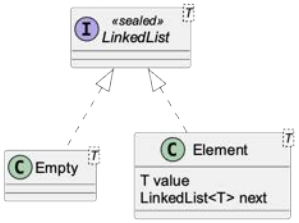
when-Clause
(Guarded Pattern)



51

```java
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



52

Unnamed Variables

```java
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]
```

53

```java
package de.sippsack.records;

import java.math.BigDecimal;
import java.util.Currency;

public record MonetaryAmount(BigDecimal value,  Currency currency) {}
```

```
➜  records javap MonetaryAmount.class
Compiled from "MonetaryAmount.java"
public final class de.sippsack.records.MonetaryAmount extends java.lang.Record {
  public de.sippsack.records.MonetaryAmount(java.math.BigDecimal, java.util.Currency);
  public final java.lang.String toString();
  public final int hashCode();
  public final boolean equals(java.lang.Object);
  public java.math.BigDecimal value();
  public java.util.Currency currency();
}
```

54

```java
MonetaryAmount tenEuro = new MonetaryAmount(BigDecimal.TEN,
Currency.getInstance("EUR"));
MonetaryAmount anotherTenEuro = new MonetaryAmount(BigDecimal.TEN);

System.out.println(tenEuro);
System.out.println(tenEuro.times(BigDecimal.TEN));

System.out.println(tenEuro.equals(anotherTenEuro)); // true
System.out.println(tenEuro.equals(new MonetaryAmount(BigDecimal.TEN,
      Currency.getInstance("USD")))); // false
```

56

feingranulare Steuerung der Vererbungshierarchie

Algebraische Datentypen

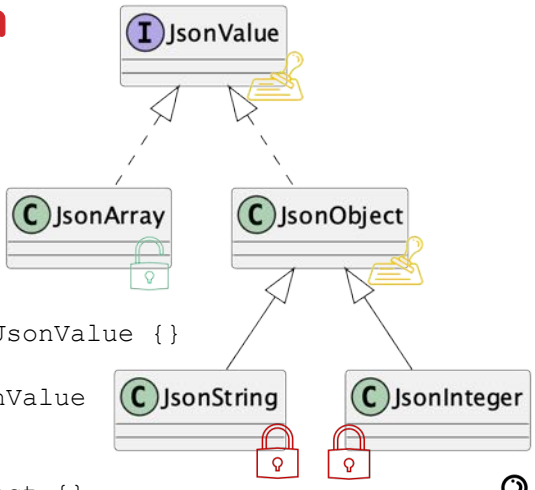
Prüfung der “Exhaustiveness” im switch

Sealed Classes

57

Flexible Klassenhierarchien

```
public sealed interface JsonValue
       permits JsonArray, JsonObject {

    non-sealed class JsonArray implements JsonValue {}

    sealed class JsonObject implements JsonValue
      permits JsonString, JsonInteger {}

    final class JsonString extends JsonObject {}

    final class JsonInteger extends JsonObject {}
}
```

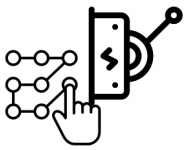


Sealed Classes

58

~~Guarded Patterns~~ When Clauses und Null-Check

```
boolean isNullOrEmptyWithSwitch( Object o ) {
  return switch(o) {
    case null -> true;
    case String s when s.isBlank() -> true;
    case String s -> false;
    case Collection c when c.isEmpty() -> true;
    default -> false;
  };
}
```



Virtual Threads

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

Millionen von Threads ...

```
Thread.Builder threadBuilder = createThreadBuilder();

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

private static Thread.Builder createThreadBuilder() {
    return Thread.ofVirtual();
    // return Thread.ofPlatform();
}
```

19

Virtual Threads

65

Aber es geht gar nicht um Millionen von Threads

- **klassisches, gleiches Thread-API** 😀
- **Vereinfachter Umgang mit einer Menge von teilweise blockierenden Aufgaben (I/O)**
  - blockiert ein Virtual Thread, wird er geparkt und der nächste ist dran
  - schnell erzeugt und leichtgewichtig -> kein Pooling notwendig, jede Aufgabe in einem neuen Thread starten
- **Virtual Threads sind nicht schneller oder effizienter (laufen auch nur auf Carrier/Platform Threads)** 😟
- **nicht für sehr rechenintensive Tasks (die nicht blockieren)**

66

Bisher: ExecutorService

```
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());
```


Structured Concurrency

```
try (var scope = new StructuredTaskScope.ShutdownOnFailure()) {
  Future<String> task1 = scope.fork(() -> { … }
  Future<String> task2 = scope.fork(() -> { … }
  Future<String> task3 = scope.fork(() -> { … }

  scope.join();
  scope.throwIfFailed();

  System.out.println(task1.get());
  System.out.println(task2.get());
  System.out.println(task3.get());
}
```

19

basiert auf
Virtual Threads

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("not valid")));
}
```


Loom – Chancen und Limitierungen

- **nicht für die Verarbeitung großer Datenmengen bzw. schwere Berechnungen**
- **Leichter für "naive" Programmierung von Threads**
- **Webcontainer können Virtual Threads einsetzen, um Speicher zu sparen**
  - mehr parallele Requests
  - aber Limitierung bleiben Ressourcen außerhalb der JVM (Datenbanken, ..)

Für wen ist es gut?

Virtual Threads unter der Haube

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

Nebenläufige Programmierung

Entwickler, die **sehr viele** und/oder **blockierende Aufgaben** implementieren müssen (einfacheres API).

71

Neue Features

Garbage Collectoren

Interna

API-Änderungen (JDK)

72

Neue Generation von Low-Latency Garbage Collectoren

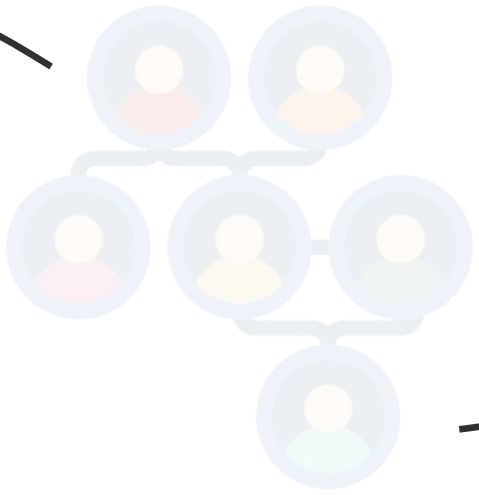
moderne Architekturen: Multi-Core + TB RAM

Ziel: kurze GC-Pausen im ms-Bereich

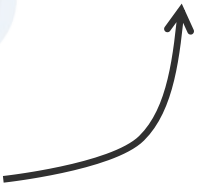
73

JEP 439: Generational ZGC

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



Young-Gen:
wird im GC-Lauf prozessiert



74

Neue Features

Garbage Collectoren

Interna

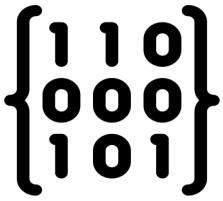
API-Änderungen (JDK)

75

14 – 21 ff.

Foreign Linker + Foreign-Memory Access API

14 – 21 ff.



Vector API

76

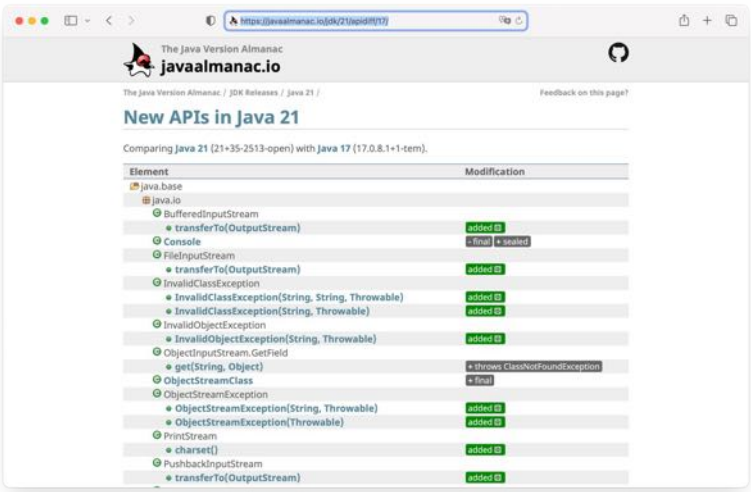
Neue Features

Garbage Collectoren

Interna

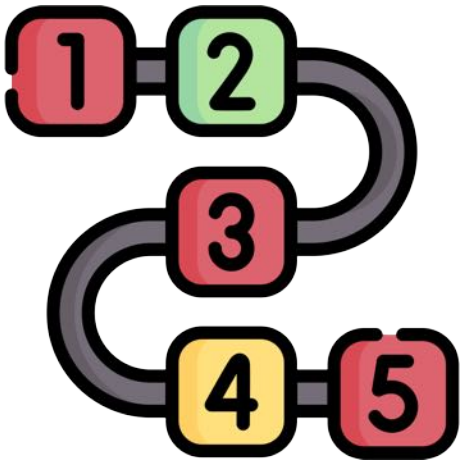
API-Änderungen (JDK)

77



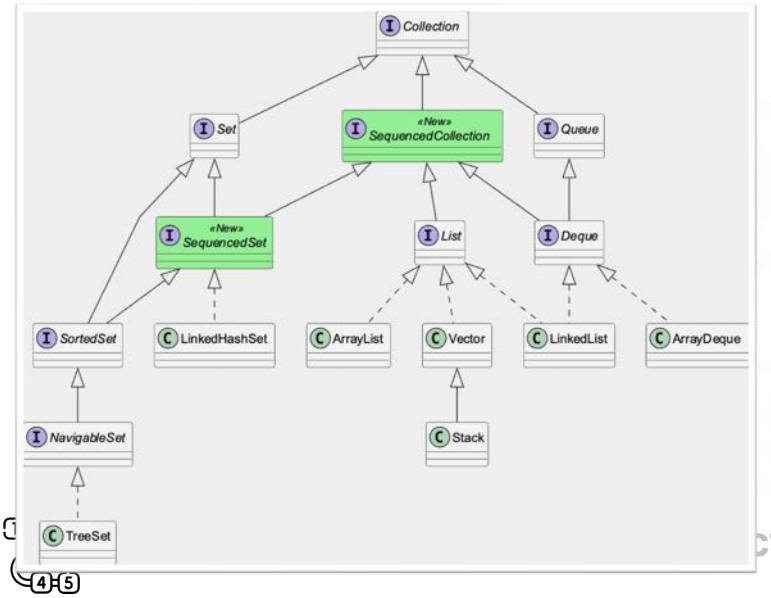
https://javaalmanac.io/jdk/21/apidiff/17/

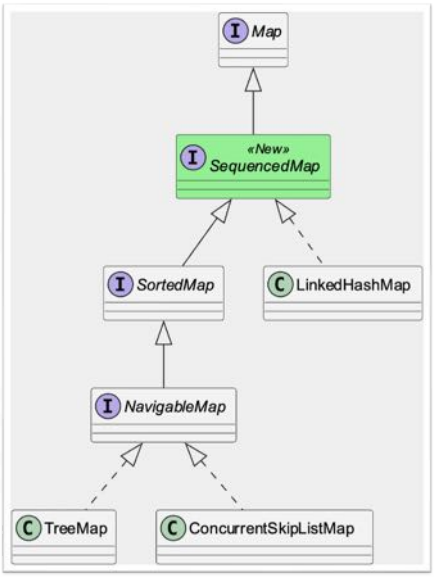
78



Sequenced Collection

79



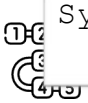


80

```
List<Integer> list = new ArrayList<>(List.of(1, 2, 3, 4, 5));
List<Integer> reversed = list.reversed();
System.out.println(reversed); // 5, 4, 3, 2, 1

list.addFirst(0);
list.addLast(6);

System.out.println(list.getFirst()); // 0
System.out.println(list.getLast()); // 6
```

81

```
SequencedMap<Integer, String > map =
     new LinkedHashMap<>(Map.of(1, "a", 2, "b"));

SequencedMap<Integer, String> reversedMap = map.reversed();
System.out.println(map); // {2=b, 1=a}
System.out.println(reversedMap); // {1=a, 2=b}

map.putFirst(0, "o");
map.putLast(3, "c");

System.out.println(map.pollFirstEntry()); // 0=o
System.out.println(reversedMap); // {3=c, 2=b, 1=a}

System.out.println(map.sequencedKeySet()
     .reversed()); // [3, 2, 1]
```



82

Aktivierung Preview + Inkubator Features

```
$ javac --enable-preview -source 19 --add-modules jdk.incubator.concurrent
StructuredConcurrency.java

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

Welche Änderungen gab es noch seit Java 17?

Features

https://openjdk.org/projects/jdk/21/

430: String Templates (Preview)
431: Sequenced Collections
439: Generational ZGC
440: Record Patterns
441: Pattern Matching for switch
442: Foreign Function & Memory API (Third Preview)
443: Unnamed Patterns and Variables (Preview)
444: Virtual Threads
445: Unnamed Classes and Instance Main Methods (Preview)
446: Scoped Values (Preview)
448: Vector API (Sixth Incubator)
449: Deprecate the Windows 32-bit x86 Port for Removal
451: Prepare to Disallow the Dynamic Loading of Agents
452: Key Encapsulation Mechanism API
453: Structured Concurrency (Preview)

JDK 21 will be a long-term support (LTS) release from most vendors. For a complete list of the JEPs integrated since the previous LTS release, JDK 17, please see here.

85

OpenJDK

Installing
Contributing
Sponsoring
Developers' Guide
Vulnerabilities
JDK GA/EA Builds
Mailing lists
Wiki · IRC
Bylaws · Census
Legal
Workshop
JEP Process
Source code
Mercurial
GitHub
Tools
Git
Jtreg harness
Groups
(overview)
Adoption
Build
Client Libraries
Compatibility & Specification Review
Compiler
Conformance
Core Libraries
Governing Board
HotSpot
IDE Tooling & Support
Internationalization
JMX
Members
Networking

JEPs in JDK 21 integrated since JDK 17

Here are all of the JEPs integrated since JDK 17, which was the previous long-term-support (LTS) release from most vendors. Preview and Incubator JEPs that were superseded by later JEPs in JDKs 18 through 21 are not included. The release in which a JEP was integrated is shown in parentheses after the JEP's title.

Additions

HotSpot JVM

422: Linux/RISC-V Port (19)

Garbage Collectors

439: Generational ZGC (21)

Serviceability

451: Prepare to Disallow the Dynamic Loading of Agents (21)

Language

441: Pattern Matching for switch (21)
440: Record Patterns (21)

Libraries

444: Virtual Threads (21)

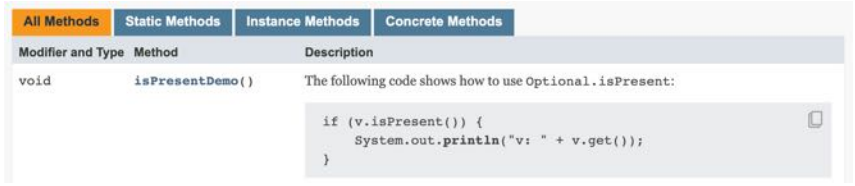
Collections

431: Sequenced Collections (21)

Cryptography

86

```
/**
 * The following code shows how to use {@code Optional.isPresent}:
 * {@snippet :
 * if (v.isPresent()) {
 *     System.out.println("v: " + v.get()); // @highlight substring="println"
 * }
 *}
 */
public void isPresentDemo() {
```



18

Code Snippets in JavaDoc

87

jwebserver

```
$ jwebserver
Binding to loopback by default. For all interfaces use "-b 0.0.0.0" or "-b ::".
Serving /home/sven and subdirectories on 127.0.0.1 port 8000
URL http://127.0.0.1:8000/
```

```
HttpServer server = SimpleFileServer.createFileServer(
  new InetSocketAddress(8888), Path.of("out"),
    OutputLevel.INFO);
server.start();
```

18

Simple Webserver

88

Auf Wiedersehen …

- 🎯 **Applet API** deprecated for Removal
- 🎯 **Security Manager** deprecated for Removal
- 🎯 Deprecate **Finalization** for Removal
- ☠️ Experimental **AOT and JIT Compiler**

90

Agenda

1 Java – still a thing?

2 Ist Java 21 ein LTS-Release?

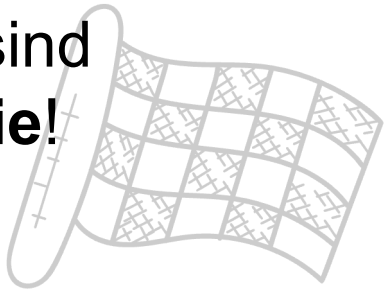
3 Viele spannende neue Features

4 **Fazit**

4

91

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



92

Every feature is not as good as a good developer can utilise it, but as bad as the average developer can misuse it.

93

Was könnte in Java 22 konkret kommen?

JEP 447: **Statements before super() (Preview)**

```
public PositiveBigInteger(long value) {
  // Invalid in Java 21
  if (value <= 0) throw new
    IllegalArgumentException("non-positive value");
  super(value);
}
```

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

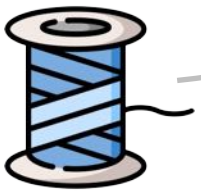
JEP draft: **Ahead of Time Compilation for the Java Virtual Machine**

94

Pattern Matching + kleine Syntax-Verbesserungen

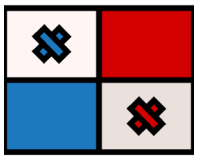
Amber

Virtual Threads



Loom

Was kommt noch?



Panama

Vector API + Foreign Function & Memory API

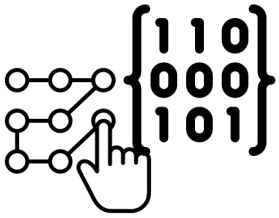
Valhalla

Value Types

95

Array Patterns

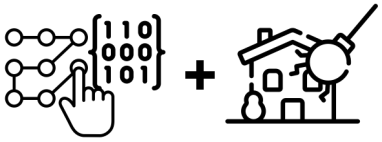
```
static void printFirstTwoStrings(Object o) {
  if (o instanceof String[] { String s1, String s2, ... }){
    System.out.println(s1 + s2);
  }
}
```



96

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);
  }
}
```

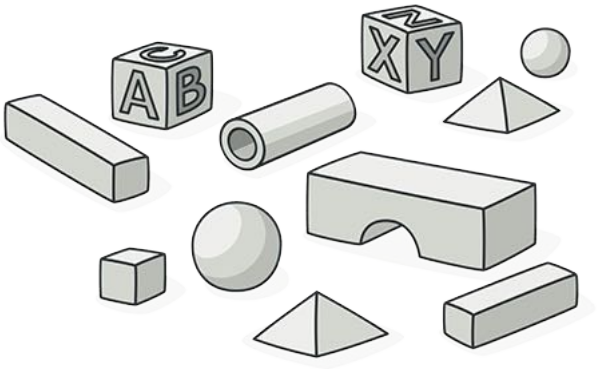


97

Record Patterns in foreach-Schleifen

98

Primitive Obsession



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

99

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

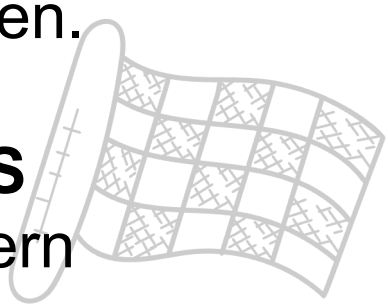
100

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
```

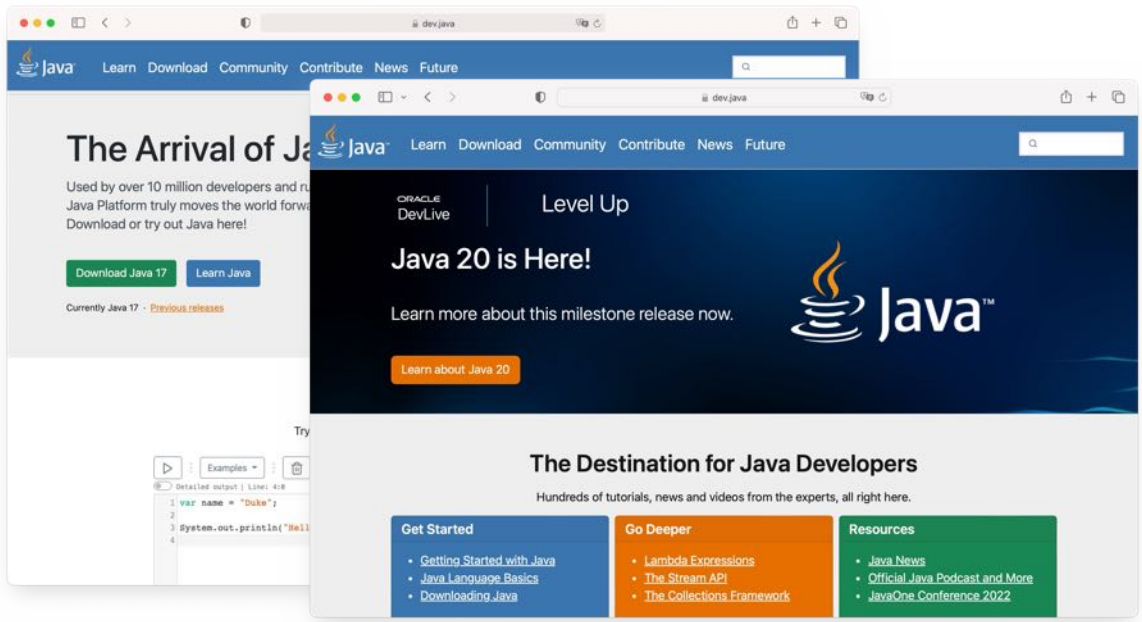
101

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



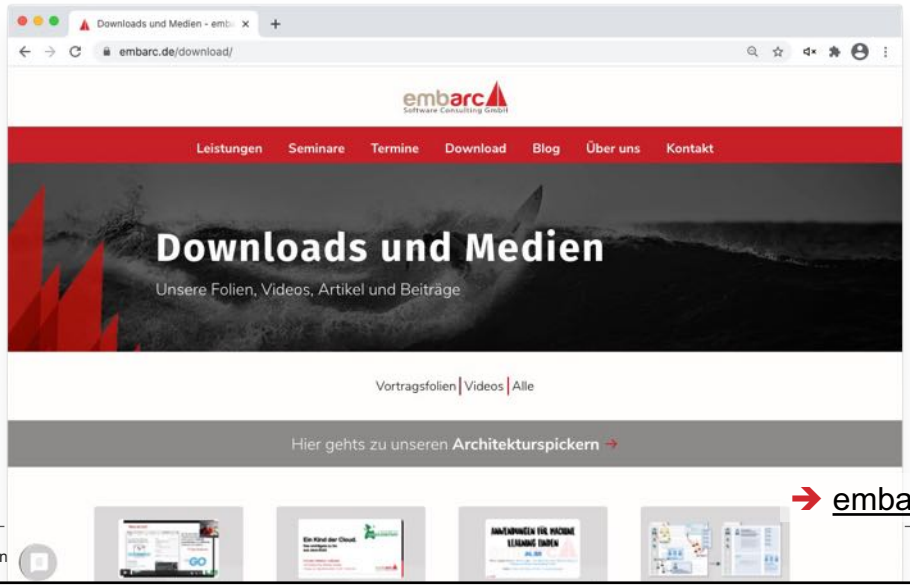
Das **nächste LTS**
Release ist nicht fern
…

102



103

Folien von heute als PDF zum Download



➔ embarc.de/download/

104

105

Vielen Dank.

Ich freue mich auf Eure Fragen!

Falk Sippach

fs@embarc.de

@sippsack

➔ xing.to/fsi