



Best of Java 18 bis 20

<https://github.com/Michaeli71/Governikus> Java Update 11 17

Michael Inden

Head of Development, freiberuflicher Buchautor und Trainer

Speaker Intro



- **Michael Inden, Jahrgang 1971**
- **Diplom-Informatiker, C.v.O. Uni Oldenburg**
- **~8 ¼ Jahre SSE bei Heidelberger Druckmaschinen AG in Kiel**
- **~6 ¾ Jahre TPL, SA bei IVU Traffic Technologies AG in Aachen**
- **~4 ¼ Jahre LSA / Trainer bei Zühlke Engineering AG in Zürich**
- **~3 Jahre TL / CTO bei Direct Mail Informatics / ASMIQ in Zürich**
- **Freiberuflicher Consultant, Trainer und Konferenz-Speaker**
- **Seit Januar 2022 Head of Development bei Adcubum in Zürich**
- **Autor und Gutachter beim dpunkt.verlag / APress**

E-Mail: michael_inden@hotmail.com

Kurse: **Bitte sprecht mich an!**

<https://github.com/Michaeli71/Best-Of-Java-11-20>



Workshop Contents



-
- **PART 6:** Neuheiten in Java 18, 19 und 20
 - **Ausblick Java 21**
-



PART 6: Neuerungen in Java 18, 19 & 20



Build-Tools und IDEs



IDE & Tool Support für Java 20



- **Eclipse: Version 2023-03 mit zusätzlichem Plugin**
- **IntelliJ: Version 2023.1**
- **Maven: 3.9.1, Compiler-Plugin: 3.11.0**
- **Gradle: 8.1.x**
- **Aktivierung von Preview-Features nötig**
 - In Dialogen
 - Im Build-Skript



Maven™

 **Gradle**

IDE & Tool Support Java 20



- Eclipse 2023-03 mit Plugin
- Aktivierung von Preview-Features nötig



Eclipse Marketplace

Select solutions to install. Press Install Now to proceed with installation.
Press the "more info" link to learn more about a solution.

Search Recent Popular Favorites Installed Giving IoT an Edge

Find: All N

Java 20 Support for Eclipse 2023-03 (4.27)

This marketplace solution provides Java 20 support for Eclipse. To use Java 20, you have the latest Eclipse release, which is... [more info](#)

by [Eclipse Foundation](#), EPL 2.0

1 Installs: 949 (796 last month)

type filter text

- > Resource Builders Coverage Java Build Path
- > Java Code Style
- > **Java Compiler**
- Javadoc Location
- > Java Editor
- > MoreUnit
- Project Natures
- Project References
- > Protocol Buffer
- Refactoring History
- Run/Debug Settings
- > SonarLint
- WikiText

Properties for Best_of_Java_11_bis_20

Java Compiler

Enable project specific settings [Configure Workspace Settings...](#)

JDK Compliance

Use compliance from execution environment 'JavaSE-20' on the [Java Build Path](#)

Compiler compliance level:

Use '--release' option

Use default compliance settings

Enable preview features for Java 20

Preview features with severity level:

Generated .class files compatibility:

Source compatibility:

Disallow identifiers called 'assert':

IDE & Tool Support



- Aktivierung von Preview-Features nötig

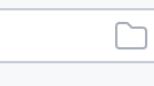
Project Structure

Project
Default settings for all modules. Configure these parameters for each module on the module page as needed.

Name: `SwitchIntroExample.java` 

SDK: `20 Oracle OpenJDK version 20`  Edit

Language level: `20 (Preview) - Record patterns (2nd preview), pattern matching for switch (4th preview)` 

Compiler output: `~/Java20Examples/out` 

Used for modules' subdirectories, Production and Test directories for the corresponding sources.

Project Settings

- Project
- Modules
- Libraries
- Facets
- Artifacts

Platform Settings

- SDKs
- Global Libraries

Problems



- Aktivierung von Preview-Features / Incubator nötig

```
sourceCompatibility=20  
targetCompatibility=20
```



```
tasks.withType(JavaCompile).configureEach {  
    options.compilerArgs += ["--enable-preview"]  
}
```

```
tasks.withType(JavaCompile).configureEach {  
    options.compilerArgs += ["--enable-preview",  
                            "--add-modules", "jdk.incubator.concurrent",  
                            "--add-modules", "jdk.incubator.vector"]  
}
```

IDE & Tool Support



- Aktivierung von Preview-Features / Incubator nötig

```
<plugins>
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.11</version>
    <configuration>
      <source>20</source>
      <target>20</target>
      <!-- Wichtig für Java Syntax-Neuerungen -->
      <compilerArgs>--enable-preview</compilerArgs>
    </configuration>
  </plugin>
</plugins>
```



```
<plugins>
  <plugin>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.11</version>
    <configuration>
      <source>20</source>
      <target>20</target>
      <compilerArgs>
        <arg>--enable-preview</arg>
        <arg>--add-modules</arg>
        <arg>jdk.incubator.concurrent</arg>
        <arg>--add-modules</arg>
        <arg>jdk.incubator.vector</arg>
      </compilerArgs>
    </configuration>
  </plugin>
<plugin>
```

IDE & Tool Support Java 20 für Vector API Incubator



Run Configurations

Create, manage, and run configurations
Run a Java application

Name: VectorApiExample

Main Arguments JRE Dependencies Source Environment Common Prototype

Program arguments:

VM arguments:

--add-modules jdk.incubator.vector

Use the -XstartOnFirstThread argument when launching with SWT

Use the -XX:+ShowCodeDetailsInExceptionMessages argument when launching

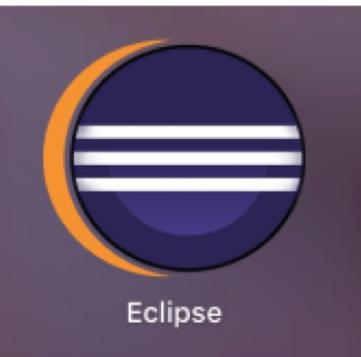
Use @argfile when launching

Working directory:

Default: \${workspace_loc:Java18Examples}

Other: _____

Show Command Line Revert Apply



IDE & Tool Support für Vector API Incubator



Screenshot of the IntelliJ IDEA Preferences dialog showing Java Compiler settings for cross-compilation.

The left sidebar shows the navigation path: **Java Compiler** > **Build, Execution, Deployment** > **Compiler** > **Java Compiler**.

Use compiler: Javac

Use '--release' option for cross-compilation (Java 9 and later)

Project bytecode version: Same as language level

Per-module bytecode version:

Module	Target bytecode version
Java20Examples	20

Javac Options

- Use compiler from module target JDK when possible
- Generate debugging info
- Report use of deprecated features
- Generate no warnings

Additional command line parameters: ('/' recommended in paths for cross-platform configurations)

```
--enable-preview --add-modules jdk.incubator.concurrent --add-modules jdk.i...
```

Override compiler parameters per-module:

Module	Compilation options
Java20Examples	--enable-preview --add-modules jdk.incubator.con...

Buttons at the bottom: **?**, **Cancel**, **Apply**, **OK**.



IDE & Tool Support für Vector API Incubator



Edit Configuration Settings

Name: VectorApiExample Store as project file

Run on: Local machine Manage targets...

Run configurations may be executed locally or on a target: for example in a Docker Container or on a remote host using SSH.

Build and run ↓ ↓ Modify options

java 20 SDK of 'Java20Exai' --add-modules jdk.incubator.vector

java20.vectorapi.VectorApiExample

Program arguments

Press ⌘ for field hints

Working directory: /Users/michaelinden/Java20Examples

Environment variables:

Separate variables with semicolon: VAR=value; VAR1=value1

Open run/debug tool window when started ×

Code Coverage Modify

Packages and classes to include in coverage data

?

Cancel Apply Run





JEPs in Java 18, 19 und 20



- JEP 400: UTF-8 by Default
- **JEP 408: Simple Web Server**
- **JEP 413: Code Snippets in Java API Documentation**
- JEP 416: Reimplement Core Reflection with Method Handles
- JEP 417: Vector API (Third Incubator)
- **JEP 418: Internet-Address Resolution SPI**
- JEP 419: Foreign Function & Memory API (Second Incubator)
- **JEP 420: Pattern Matching for switch (Second Preview)**
- JEP 421: Deprecate Finalization for Removal

- **JEP 405: Record Patterns (Preview)**
- JEP 422: Linux/RISC-V Port
- JEP 424: Foreign Function & Memory API (Preview)
- JEP 425: Virtual Threads (Preview)
- JEP 426: Vector API (Fourth Incubator)
- **JEP 427: Pattern Matching for switch (Third Preview)**
- JEP 428: Structured Concurrency (Incubator)

18

19



- **JEP 405: Record Patterns (Preview)**
 - JEP 422: Linux/RISC-V Port
 - JEP 424: Foreign Function & Memory API (Preview)
 - JEP 425: Virtual Threads (Preview)
 - JEP 426: Vector API (Fourth Incubator)
 - **JEP 427: Pattern Matching for switch (Third Preview)**
 - JEP 428: Structured Concurrency (Incubator)
-
- JEP 429: Scoped Values (Incubator)
 - **JEP 432: Record Patterns (Second Preview)**
 - **JEP 433: Pattern Matching for switch (Fourth Preview)**
 - JEP 434: Foreign Function & Memory API (Second Preview)
 - **JEP 436: Virtual Threads (Second Preview)**
 - **JEP 437: Structured Concurrency (Second Incubator)**
 - **JEP 438: Vector API (Fifth Incubator)**

19

20



JEP 408: Simple Web Server





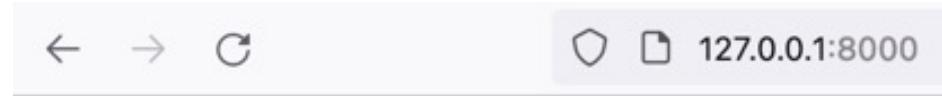
- Ziel bei diesem JEP war es, einen minimalistischen Webserver ohne viel Konfiguration ins JDK zu integrieren, der lediglich statische Dateien bereitstellen kann. Dies erleichtert die Erstellung von Prototypen, das File Sharing oder kann auch für einfache Tests hilfreich sein.

```
michaelinden@MBP-von-Michael ~ % jwebserver
Binding to loopback by default. For all interfaces use "-b 0.0.0.0" or "-b ::".
Serving /Users/michaelinden and subdirectories on 127.0.0.1 port 8000
URL http://127.0.0.1:8000/
```

```
jshell> import com.sun.net.httpserver.SimpleFileServer;
...> import com.sun.net.httpserver.SimpleFileServer.OutputLevel;

jshell> var server = SimpleFileServer.createFileServer(new
InetSocketAddress(8000), Path.of("/Users/michaelinden"), OutputLevel.VERBOSE)
server ==> sun.net.httpserver.HttpServerImpl@6659c656

jshell> server.start()
```



Directory listing for /

- [Music/](#)
- [JavaDateAndTimeOverview_RO.astar](#)
- [spring-boot-test-workspace/](#)
- [JDK_SET.txt](#)
- [spacerocks.dat](#)
- [java8-workspace/](#)
- [h2](#)
- [spring-boot-persistence-data-workspace/](#)
- [Pictures/](#)
- [zshrc.textClipping](#)
- [JavaDateAndTimeOvervie_IFs.png](#)
- [Desktop/](#)
- [Library/](#)
- [eclipse-workspace/](#)
- [mongodb-macos-x86_64-5.0.3/](#)
- [spring-framework-intro-workspace/](#)
- [Sites/](#)
- [splash.png](#)
- [my.properties](#)
- [PycharmProjects/](#)



JEP 413: Code Snippets in Java API Documentation



JEP 413: Code Snippets in Java API Documentation



- Bislang gibt es keinen Standard, um Codefragmente in die mit Javadoc generierte HTML-Dokumentation aufzunehmen. Im Rahmen dieses JEPs wird das Inline-Tag @snippet eingeführt. Damit lassen sich Codefragmente in einen Javadoc-Kommentar einfügen:

```
/**  
 * The following code shows how to use {@code Optional.isPresent}:  
 * {@snippet :  
 * if (optValue.isPresent())  
 * {  
 *     System.out.println("value: " + optValue.get());  
 * }  
 * }  
 */  
  
public static void method1(String[] args) {  
    //  
    //  
    //  
    //  
}  
}
```

 **void java18.misc.JavaDocExample.method1(String[] args)**

The following code shows how to use `Optional.isPresent`:

```
if (optValue.isPresent())  
{  
    System.out.println("value: " + optValue.get());  
}
```



- **Hervorherbungen**

```
/**  
 * The following code shows how to use {@code Optional.isPresent} and  
 * {@code Optional.get} in combination  
 *  
 * {@snippet :  
 * if (optValue.isPresent()) // @highlight substring="isPresent"  
 * {  
 *     System.out.println("value: " + optValue.get()); // @highlight substring="get"  
 * } }  
 */  
  
public static void newJavaDocExample(String[] args) {  
    //  
    //  
    //  
    //  
}  
}
```

 **void java18.misc.JavaDocExample.newJavaDocExample(String[] args)**

The following code shows how to use `Optional.isPresent` and `Optional.get` in combination

```
if (optValue.isPresent())  
{  
    System.out.println("value: " + optValue.get());  
}
```

Parameters:
args



JEP 418: Internet-Address Resolution SPI



JEP 418: Internet-Address Resolution SPI



- Im Rahmen dieses JEP wurde ein SPI (Service Provider Interface) für die Auflösung von Host- und Namensadressen implementiert.
- Die Adressauflösung verwendet bislang eine hartcodierte Umsetzung innerhalb der Klasse `java.net.InetAddress`.

```
import java.net.InetAddress;
import java.net.UnknownHostException;
import java.util.Arrays;

public class InetAddressExample
{
    public static void main(String[] args) throws UnknownHostException
    {
        InetAddress[] addresses = InetAddress.getAllByName("www.dpunkt.de");
        System.out.println("addresses = " + Arrays.toString(addresses));
    }
}
```

JEP 418: Internet-Address Resolution SPI



```
public class OwnInetAddressResolver implements InetAddressResolver
{
    @Override
    public Stream<InetAddress> lookupByName(String host, LookupPolicy lookupPolicy)
        throws UnknownHostException
    {
        if (host.equals("www.dpunkt.de"))
            return Stream.of(InetAddress.getByAddress(new byte[] { 127, 0, 0, 1 }));

        throw new UnsupportedOperationException();
    }

    @Override
    public String lookupByAddress(byte[] addr)
    {
        throw new UnsupportedOperationException();
    }
}
```

JEP 418: Internet-Address Resolution SPI



```
import java.net.spi.InetAddressResolver;
import java.net.spi.InetAddressResolverProvider;

public class OwnInetAddressResolverProvider extends InetAddressResolverProvider
{
    @Override
    public InetAddressResolver get(Configuration configuration) {
        return new OwnInetAddressResolver();
    }

    @Override
    public String name() {
        return "Own Internet Address Resolver Provider";
    }
}
```





JEP 405: Record Patterns (Preview in Java 19)





- Basis für diesen JEP ist das Pattern Matching für instanceof aus Java 16:

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

static void printCoordinateInfo(Object obj)
{
    if (obj instanceof Point point)
    {
        int x = point.x();
        int y = point.y();

        System.out.println("x: %d y: %d, sum: %d".formatted(x, y, x + y));
    }
}
```

- Zwar ist das oftmals schon praktisch, aber man muss noch teilweise umständlich auf die einzelnen Bestandteile zugreifen.



- Ziel ist es, Records deklarativ in ihre Bestandteile zerlegen und zugreifen zu können.

```
static void printCoordinateInfo(Object obj)
{
    if (obj instanceof Point point)
    {
        int x = point.x();
        int y = point.y();
        System.out.println("x: %d y: %d, sum: %d".formatted(x, y, x + y));
    }
}
```

- =>

```
static void printCoordinateInfoNew(Object obj)
{
    if (obj instanceof Point(int x, int y))
        System.out.println("x: %d y: %d, sum: %d".formatted(x, y, x + y));
}
```



- Record Patterns können verschachtelt werden, um eine deklarative, leistungsfähige und kombinierbare Form der Datennavigation und -verarbeitung zu ermöglichen.

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

enum Color { RED, GREEN, BLUE }
record ColoredPoint(Point p, Color c) {}
record Rectangle(ColoredPoint upperLeft, ColoredPoint lowerRight) {}
```

```
static void printColorOfUpperLeftPoint(Rectangle rect)
{
    if (rect instanceof Rectangle(ColoredPoint(Point p, Color c),
                                  ColoredPoint lr))
    {
        System.out.println(c);
    }
}
```



DEMO & Hands on

Jep405_RecordPatternsExample.java

Jep405_InstanceofRecordMatchingAdvanced.java



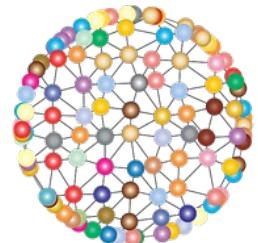
- Record Patterns können für Eleganz sorgen:

```
record Person(String name, int age, Boolean hasDrivingLicense) { }

boolean isAllowedToDrive(Object obj) {
    if (obj instanceof Person person) {
        return person.age() >= 18 && person.hasDrivingLicense();
    }
    return false;
}

boolean isAllowedToDrive(Object obj) {
    if (obj instanceof Person(String name, int age, Boolean hasDrivingLicense)) {
        return age >= 18 && hasDrivingLicense;
    }
    return false;
}
```

- Bitte aber immer auch gutes OO-Design im Hinterkopf haben!



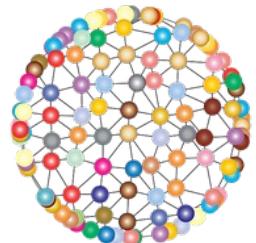
**Wie kann man das Ganze noch
eleganter gestalten?**



- **Sauberer OO-Design würde die Methode im Record selbst definieren:**

```
record Person(String name, int age, Boolean hasDrivingLicense) {  
    boolean isAllowedToDrive() {  
        return age() >= 18 && hasDrivingLicense();  
    }  
}
```

In dem vorherigen Beispiel dient der Zugriff auf die Attribute lediglich dazu, das Pattern Matching zu illustrieren.



**Wo können Record Patterns
ihre Stärke ausspielen?**



- Nehmen wir einmal folgende Records als Datenmodell an:

```
record Person(String firstname, String lastname, LocalDate birthday) {  
}
```

```
record Phone(String areaCode, String number) {  
}
```

```
record City(String name, String country, String languageCode) {  
}
```

```
record FlightReservation(Person person,  
                        Phone phoneNumber,  
                        City from,  
                        City destination) {  
}
```



- In Legacy-Code findet man tief verschachtelte Abfragen wie diese:

```
boolean checkAgeAndDestinationLanguageCorrectOld(Object obj)
{
    if (obj instanceof FlightReservation reservation)
    {
        if (reservation.person() != null)
        {
            Person person = reservation.person();

            if (reservation.destination() != null)
            {
                LocalDate birthday = person.birthday();
                String languageCode = destination.languageCode();

                if (birthday != null && languageCode != null)
                {
                    long years = ChronoUnit.YEARS.between(LocalDate.now(), birthday);
                    return years >= 18 && List.of("EN", "DE", "FR").contains(languageCode);
                }
            }
        }
    }
    return false;
}
```

Jep405_FlighReservationExample.java



- Mit verschachtelten Record Patterns schreiben wir die obige Verschachtelung der ifs elegant und viel verständlicher wie folgt:

```
boolean checkAgeAndDestinationLanguageCorrectNew(Object obj)
{
    if (obj instanceof FlightReservation(
        Person(String firstname, String lastname, LocalDate birthday),
        Phone phoneNumber, City from,
        City(String name, String country, String languageCode)))
    {
        if (birthday != null && languageCode != null)
        {
            long years = ChronoUnit.YEARS.between(LocalDate.now(), birthday);
            return years >= 18 &&
                List.of("EN", "DE", "FR").contains(languageCode);
        }
    }
    return false;
}
```

JEP 405: Record Patterns



```
boolean checkAgeAndDestinationLanguageCorrectNew(Object obj)
{
    if (obj instanceof FlightReservation(
        Person(String firstname, String lastname, LocalDate birthday),
        Phone phoneNumber, City from,
        City(String name, String country, String languageCode)))
    {
        if (birthday != null && languageCode != null)
        {
            long years = ChronoUnit.YEARS.between(LocalDate.now(), birthday);
            return years >= 18 &&
                List.of("EN", "DE", "FR").contains(languageCode);
        }
    }
    return false;
}
```

- Die Prüfung mit `instanceof` schlägt automatisch fehl, falls eine der Record-Komponenten `null` ist, also hier `Person` oder `City` (destination).
- Lediglich die Attribute werden so nicht abgesichert und sind ggf. auf `null` zu prüfen.
- Wenn man sich jedoch den guten Stil angewöhnt, `null` als Wert von Parametern bei Aufrufen zu vermeiden, kann man sogar darauf verzichten.



Insgesamt bietet Java 19 die folgenden drei Möglichkeiten, Pattern Matching für Records durchzuführen:

- 1) Pattern Matching – Zugriff über Variable und Methoden
- 2) Record Pattern – Dekomposition in Einzelbestandteile
- 3) Named Record Pattern – Kombination aus 1) und 2)

```
record StringIntPair(String name, int value) {}

Object obj = new StringIntPair("Michael", 52);

// 1. Pattern Matching
if (obj instanceof StringIntPair pair) {
    System.out.println("object is a StringIntPair, name = " +
                       pair.name() + ", value = " + pair.value());
}

// 2. Record Pattern
if (obj instanceof StringIntPair(String name, int value)) {
    System.out.println("object is a StringIntPair, " +
                       "name = " + name + ", value = " + value);
}

// 3. Named Record Pattern
if (obj instanceof StringIntPair(String name, int value) pair) {
    System.out.println("object is a StringIntPair, name = " +
                       pair.name() + ", value = " + pair.value() +
                       "// name = " + name + ", value = " + value);
}
```



JEP 432: Record Patterns (Second Preview in Java 20)





Mit Java 20 werden folgende drei Verbesserungen bei Record Patterns umgesetzt:

- 1. Die Inferenz von Typargumenten und generische Record Patterns wurde verbessert.**
- 2. Record Patterns lassen sich nun in for-each-Schleifen nutzen.**
- 3. Named Record Patterns wurden entfernt.**



- Wenn Generics in Record Patterns genutzt werden, erfordern Abfragen in Java 19 entsprechende Typangaben.

```
void handleContainerOld(final Container<String> container)
{
    if (container instanceof Tuple<String>(var s1, var s2))
    {
        System.out.println("Tuple: " + s1 + ", " + s2);
    }
    else if (container instanceof Triple<String>(var s1, var s2, var s3))
    {
        System.out.println("Triple: " + s1 + ", " + s2 + ", " + s3);
    }
}

interface Container<T> {}
record Tuple<T>(T t1, T t2) implements Container<T> {}
record Triple<T>(T t1, T t2, T t3) implements Container<T> {}
```



- Verbesserte Typinferenz erlaubt eine übersichtlichere Schreibweise ohne Typangabe in <>:

```
void handleContainerNew(Container<String> container)
{
    if (container instanceof Tuple(var s1, var s2))
    {
        System.out.println("Tuple: " + s1 + ", " + s2);
    }
    else if (container instanceof Triple(var s1, var s2, var s3))
    {
        System.out.println("Triple: " + s1 + ", " + s2 + ", " + s3);
    }
}
```

- Das **wirkt** allerdings ein wenig wie die **Raw Types** von Collections.
- Während die alte Syntax problemlos in Java 20 funktioniert, führt die neue Syntax unter Java 19 zur Fehlermeldung «raw deconstruction patterns are not allowed»



- Insbesondere bei Verschachtelungen war die alte Schreibweise mit Typangabe in <> schwierig lesbar:

```
void nestedOld(Container<Tuple<String>> container)
{
    if (container instanceof Tuple<Tuple<String>>(Tuple(var s1, var s2),
                                                Tuple(var s3, var s4)))
    {
        System.out.println("String " + String.join("/", 
                                         List.of(s1, s2, s3, s4)));
    }
    else
    {
        System.out.println("Container " + container);
    }
}
```



- **Verbesserte Typinferenz erlaubt eine übersichtlichere Schreibweise ohne Typangabe in <>:**

```
void nestedNew(Container<Tuple<String>> container)
{
    if (container instanceof Tuple(Tuple(var s1, var s2),
                                  Tuple(var s3, var s4)))
    {
        System.out.println("String " + String.join("/", 
                                         List.of(s1, s2, s3, s4)));
    }
    else
    {
        System.out.println("Container " + container);
    }
}
```

- **Noch besser wäre nach meinem Verständnis folgende Schreibweise:**

```
if (container instanceof Tuple<>(Tuple(var s1, var s2),
                                  Tuple(var s3, var s4)))
```



- Probieren wir die zuvor definierten Methoden einmal aus:

```
jshell> handleContainerNew(new Tuple<>("ONE", "TWO"));  
Tuple: ONE, TWO
```

```
jshell> handleContainerNew(new Triple<>("ONE", "TWO", "THREE"));  
Triple: ONE, TWO, THREE
```

```
jshell> nestedNew(new Tuple<>(new Tuple<>("1-1", "1-2"),  
...>                                new Tuple<>("2-1", "2-2")))  
String 1-1/1-2/2-1/2-2
```

```
jshell> nestedNew(new Triple<>(new Tuple<>("1-1", "1-2"),  
...>                                new Tuple<>("2-1", "2-2"),  
...>                                new Tuple<>("3-1", "3-2")))  
Container Triple[t1=Tuple[t1=1-1, t2=1-2], t2=Tuple[t1=2-1, t2=2-2],  
t3=Tuple[t1=3-1, t2=3-2]]
```



- Gegeben seien folgende Ausgangsdaten:

```
record City(String name, int inhabitants) {}

var cities = List.of(new City("Zürich", 400_000),
    new City("Kiel", 265_000),
    new City("Köln", 1_000_000),
    new City("Berlin", 3_500_000));
```

- Seit Java 20 lassen sich Record Pattern in einer for-each-Schleife angeben. Dadurch wird es möglich, direkt auf die Attribute zuzugreifen (genau wie bei instanceof und switch):

```
for (City(var name, var inhabitants) : cities)
{
    System.out.println(name + " has " + inhabitants + " inhabitants");
}
```

- null-Werte im Datenbestand lösen eine MatchException aus



- Nehmen wir an, RP sei ein Record Pattern und wir durchlaufen eine for-each-Schleife mit einem Block STMT-BLOCK wie folgt:

```
for (RP : elements) {  
    STMT-BLOCK  
}
```

- Das Ganze ist äquivalent zur folgenden for-each-Schleife, die ein switch mit einem Record Pattern sowie dem neuen Spezialfall case null nutzt:

```
for (var tmp : elements) {  
    switch (tmp) {  
        case null -> throw new MatchException(new NullPointerException());  
        case RP -> STMT-BLOCK;  
    }  
}
```

- Erkenntnis: Damit Record Patterns in for-each-Schleifen funktionieren, müssen alle Elemente zuweisungskompatibel zu dem Record Pattern sein.



- Keine Unterstützung für Named Record Patterns mehr

```
if (obj instanceof Person(var firstname, var lastname,  
                         var dateOfBirth, var eyeColor) person)
```

- Warum? Diese Schreibweise führt zu der «Inkonsistenz» / Doppeldeutigkeit, dass man über mehrere Wege auf die Variablen zugreifen kann, etwa auf den Vornamen wie folgt:

- `person.firstname()` – mit der benannten Variable und der Accessor-Methode
- `firstname` – auf Basis der Dekonstruktion

- Für mehr Stringenz ist dies mit Java 20 nicht mehr erlaubt und führt zu einem Kompilierfehler. Es wird nur noch folgende syntaktisch klarere Variante unterstützt:

```
if (obj instanceof Person(var firstname, var lastname,  
                         var dateOfBirth, var eyeColor))
```



JEP 420: Pattern Matching bei switch

(Second Preview in Java 18)



JEP 420: Pattern Matching bei switch



- JEP 420 führt zu zwei Änderungen bei der Auswertung der case innerhalb switch beim Kompilieren:
 - zum einen ändert sich die sogenannte Dominanzprüfung,
 - zum anderen wurde die Vollständigkeitsanalyse korrigiert.
- Allerdings ist das Ganze wieder in Form eines Preview Features umgesetzt und zum Nachvollziehen müssen Sie diese geeignet aktivieren, etwa wie folgt in der JShell:

```
michaelinden@MBP-von-Michael ~ % jshell --enable-preview
| Welcome to JShell -- Version 18-ea
| For an introduction type: /help intro
```



- **Problemfeld:** Es können mehrere Pattern auf eine Eingabe matchen.

```
public static void main(String[] args) {
    multiMatch("Python");
    multiMatch(null);
}

static void multiMatch(Object obj) {
    switch (obj) {
        case null -> System.out.println("null");
        case String s && s.length() > 5 -> System.out.println(s.toUpperCase());
        case String s
        -> System.out.println(s.toLowerCase());
        case Integer i
        -> System.out.println(i * i);
        default -> {}
    }
}
```

- Dasjenige, das «am allgemeingültigsten» passt, wird dominierendes Pattern genannt.
- Im Beispiel dominiert das kürzere Pattern `String s` das längere davor angegebene.



- Problematisch wird das Ganze, wenn die Reihenfolge der Patterns vertauscht wird:

```
static void dominanceExample(Object obj)
{
    switch (obj)
    {
        case null -> System.out.println("null");
        case String str -> System.out.println(str.toLowerCase());
        case String str && str.length() > 5 -> System.out.println(str.strip());
        case Integer i -> System.out.println(i);
        default -> { }
    }
}
```

A screenshot of an IDE showing Java code. A tooltip is displayed over the third case statement. The tooltip contains the following text:
Label is dominated by a preceding case label 'String str'
Move switch branch 'String str && str.length() > 5' before 'String str'
© java.lang.String

- Die Dominanzprüfung deckt das Problem auf und führt seit Java 18 und «älterem» Java 17.0.6 zu einem Kompilierfehler, da das zweite case de facto unreachable code ist.
- Mit den ersten Java 17-Versionen gab das noch keinen Fehler!



- Probleme mit Konstanten (wurde mit Java 17 nicht entdeckt)

```
static void dominanceExampleWithConstant(Object obj) {  
    switch (obj.toString()) {  
        case String s && s.length() > 5 -> System.out.println(s.toUpperCase());  
        case "Sophie" -> System.out.println("My lovely daughter");  
        default -> System.out.println("FALLBACK");  
    }  
}
```

This case label is dominated by one of the preceding case label
Press 'F2' for focus

- Korrektur, sodass das speziellste Pattern ganz oben ist:

```
static void dominanceExampleWithConstant(Object obj) {  
    switch (obj.toString()) {  
        case "Sophie" -> System.out.println("My lovely daughter");  
        case String s && s.length() > 5 -> System.out.println(s.toUpperCase());  
        default -> System.out.println("FALLBACK");  
    }  
}
```



- Betrachten wir ein Beispiel zur Abfrage verschiedener Spezialfälle eines Integer:

- zunächst einiger fixer Werte,
- dann dem positiven Wertebereich und
- danach dem verbliebenen Rest:

```
Integer value = 4711;

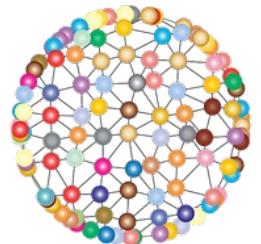
switch (value) {
    case -1, 0, 1 -> System.out.println("Handle special cases -1, 0 or 1");
    case Integer i && i > 1 ->
        System.out.println("Handle positive integer cases i > 1");
    case Integer i -> System.out.println("Handle all the remaining integers");
}
```

JEP 420: Pattern Matching bei switch: Dominanzprüfung



```
Integer value = 4711;
switch (value) {
    case Integer i && i > 1 -> System.out.println("Handle positive integer cas
    case -1, 0, 1 -> System.out.println("Handle special cases -1, 0 or 1");
    case Integer i : Label is dominated by a preceding case label 'Integer i && i > 1'
}
Move switch branch '1' before 'Integer i && i > 1' ⌂↑↔ More actions... ⌂↔
```

```
switch (value) {
    case Integer i -> System.out.println("Handle all the remaining integers");
    case Integer i && i > 1 -> System.out.println("Handle positive integer cas
    case -1, 0, 1 -> System.out.println("Handle special cases -1, 0 or 1");
}
Label is dominated by a preceding case label 'Integer i'
Move switch branch '1' before 'Integer i' ⌂↑↔ More actions... ⌂↔
```



**Was gilt es bei der
Dominanzprüfung zu beachten?**

JEP 420: Pattern Matching bei switch – Spezialfälle I



```
static void testTriangleAndString(Object obj)
{
    switch (obj)
    {
        case Triangle t && t.calculateArea() > 7000 ->
            System.out.println("Large triangle");
        case String str && str.startsWith("INFO") && str.contains("SPECIAL") ->
            System.out.println("a very special info");
        case String str && str.startsWith("INFO") ->
            System.out.println("just an info");
        // not detected ...
        case String str && str.startsWith("INFO") && str.contains("SPECIAL") ->
            System.out.println("a very special info"); DUPLIKATE IN DER ABFRAGE
        default ->
            System.out.println("Something else: " + obj);
    }
}
```

JEP 420: Pattern Matching bei switch – Spezialfälle II



```
static void testTriangleAndString(Object obj)
{
    switch (obj)
    {
        case Triangle t && t.calculateArea() > 7000 ->
            System.out.println("Large triangle");
        // case String str && str.startsWith("INFO") && str.contains("SPECIAL") ->
        //     System.out.println("a very special info");
        case String str && str.startsWith("INFO") ->
            System.out.println("just an info");
        // not detected ...
        case String str && str.startsWith("INFO") && str.contains("SPECIAL") ->
            System.out.println("a very special info");
        default ->
            System.out.println("Something else: " + obj);
    }
}
```

SPEZIALFALL IN DER ABFRAGE
=> DOMINANZ



- **Vollständigkeitsanalyse = Prüfung, ob alle möglichen Pfade von den cases im switch abgedeckt werden**
- In früheren Java 17 Versionen kam es fälschlicherweise zu der Fehlermeldung «switch statement does not cover all possible input values»

```
static void performAction(BaseOp op) {  
    switch (op) {  
        case Add a -> System.out.println(a);  
        case Sub s -> System.out.println(s);  
        // default -> System.out.println("FALLBACK")  
    }  
}
```



- Mit Java 17.0.6 ist das nicht mehr der Fall und der Fix wurde backgeportet.



- Java 18 bringt einen Bug Fix im Bereich der Vollständigkeitsanalyse, sodass folgender Sourcecode ohne Fehler kompiliert:

```
static sealed abstract class BaseOp permits Add, Sub {  
}  
  
static final class Add extends BaseOp {  
}  
  
static final class Sub extends BaseOp {  
}  
  
static void performAction(BaseOp op) {  
    switch (op) {  
        case Add a -> System.out.println(a);  
        case Sub s -> System.out.println(s);  
    }  
}
```



DEMO & Hands on

`SwitchPreviewExample.java`
`SwitchSpecialCasesExample.java`
`SwitchDominanceExample.java`
`SwitchCompletenessExample.java`



JEP 427: Pattern Matching for switch

(Third Preview in Java 19)



JEP 427: Pattern Matching bei switch



- Dieser JEP hat bereits zwei Vorgänger, die signifikante Verbesserungen bei switch mit sich brachten.
- In diesem JEP geht es um ein paar Feinheiten, und zwar die Art und Weise, wie man weitere Abfragen, sogenannte Guarded Patterns, in switch angeben kann.
- Bislang intuitiv und wie von if bekannt mit &&:

```
case String str && str.startsWith("INFO") -> System.out.println("just an info");
```

- Mit Java 19 wurde dafür das Schlüsselwort when eingeführt:

```
case String str when str.startsWith("INFO") -> System.out.println("just an info");
```

JEP 427: Pattern Matching bei switch



```
interface Shape {}

record Rectangle() implements Shape {}
record Triangle() implements Shape { int calculateArea() { return 7271; } }

static void testTriangleAndString(Object obj)
{
    switch (obj)
    {
        case Triangle t when t.calculateArea() > 7000 ->
            System.out.println("Large triangle");
        case String str when str.startsWith("INFO") ->
            System.out.println("just an info");
        default -> System.out.println("Something else: " + obj);
    }
}
```

JEP 427: Pattern Matching bei switch mit Record Patterns



```
record Pos3D(int x, int y, int z) { }

enum RgbColor {RED, GREEN, BLUE}

static void recordPatternsAndMatching(Object obj) {
    switch (obj) {
        case RgbColor color when color == RgbColor.RED ->
            System.out.println("RED WARNING");

        case Pos3D pos when pos.z() == 0 ->
            System.out.println("Record: " + pos);

        case Pos3D(int x, int y, int z) when y > 0 ->
            System.out.println("Record decomposed: " + x + ", " + y + ", " + z);

        default -> System.out.println("Something else");
    }
}

SwitchWhen.java
```



DEMO & Hands on

SwitchWhen.java



JEP 433: Pattern Matching for switch

(Forth Preview in Java 20)



JEP 433: Pattern Matching bei switch



JEP 433 zielt darauf ab, switch leistungsfähiger sowie einfacher lesbar und wartbar zu machen. Zu den wichtigsten Änderungen in Java 20 gehören die folgenden zwei:

1. die Möglichkeit, die Typargumente für generische Patterns und Record-Patterns in switch abzuleiten.
2. Bei der vollumfänglichen, auch erschöpfenden, Beschreibung aller Fälle in einem switch wird jetzt eine MatchException und nicht mehr ein IncompatibleClassChangeError ausgelöst, wenn zur Laufzeit kein case aus dem switch zutrifft.*

*Dieses Problem kann jedoch nur dann vorkommen, wenn Teile der Applikation unabhängig voneinander kompiliert werden und später dann ein Enum oder eine Klassenhierarchie erweitert und die nutzende Klasse nicht erneut kompiliert wird.

JEP 433: Pattern Matching bei switch



- Mit Java 19 musste man beim Pattern Matching noch die Typen in <> angeben:

```
record MyPair<T1, T2>(T1 first, T2 second) { }

static void recordInferenceJdk19(MyPair<String, Integer> pair)
{
    switch (pair) {
        case MyPair<String, Integer>(var text, var count)
            when text.contains("Michael") ->
                System.out.println(text + " is " + count + " years old");
        case MyPair<String, Integer>(var text, var count)
            when count > 5 && count < 10 ->
                System.out.println("repeated " + text.repeat(count));
        case MyPair<String, Integer>(var text, var count) ->
                System.out.println(text + count);
        default -> System.out.println("NOT HANDLED");
    }
}
```

JEP 433: Pattern Matching bei switch



- Analog zu der Änderung bei instanceof kann man mit Java 20 auch in switch auf die konkreten Typangaben für generische Record Patterns verzichten (nur ab JDK 20.0.1*)

```
static void recordInferenceJdk20(MyPair<String, Integer> pair)
{
    switch (pair)
    {
        // var geht hier nicht, wenn man auf die typspezifischen
        // Methode zugreifen möchte,
        case MyPair(String text, var count) when text.contains("Michael") ->
            System.out.println(text + " is " + count + " years old");
        case MyPair(String text, Integer count) when count > 5 && count < 10 ->
            System.out.println("repeated " + text.repeat(count));
        case MyPair(var text, var count) -> System.out.println(text + count);
        default -> System.out.println("NOT HANDLED");
    }
}
```

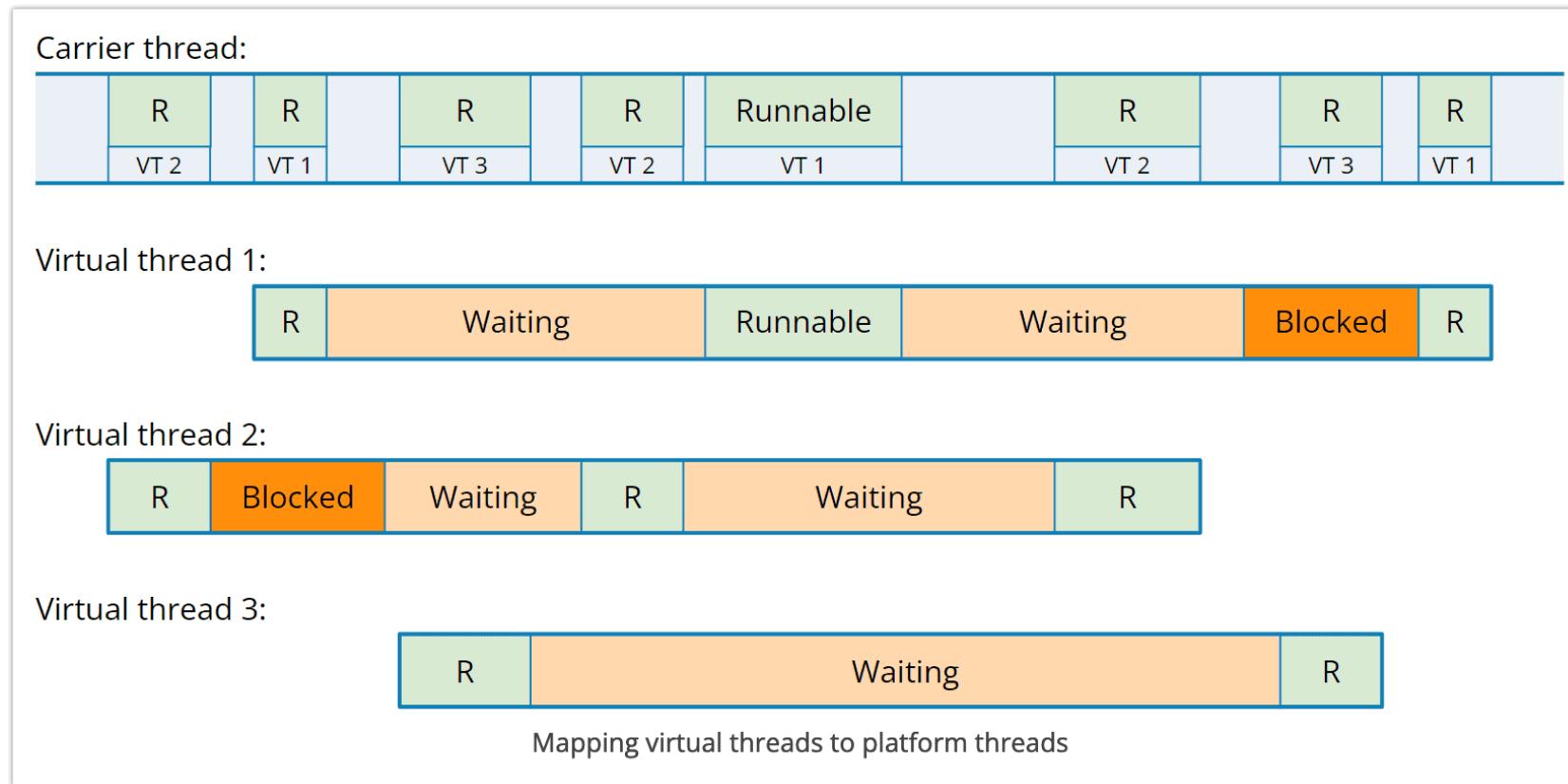


JEP 436: Virtual Threads (Second Preview in Java 20)





- Dieses Preview-Feature führt das Konzept leichtgewichtiger virtueller Threads ein.
- Virtuelle Threads «fühlen» sich wie normale Threads an, werden aber nicht 1:1 auf Betriebssystem-Threads abgebildet.





- Dieses Preview-Feature führt das Konzept **leichtgewichtiger virtueller Threads ein, die nicht direkt auf Threads des Betriebssystems abgebildet werden.**
- Besser noch: Bereits vorhandener Code, der das bisherige Thread-API verwendet, lässt sich mit minimalen Änderungen auf **virtuelle Threads umstellen.**
- Mit **virtuellen Threads** kann man im Bereich der Serverprogrammierung wieder mit jeweils einem separaten Thread pro Anfrage arbeiten und leichtgewichtiger einen **asynchronen Programmierstil unterstützen.**
- Über Factory-Methoden wie `newVirtualThreadPerTaskExecutor()` kann man wählen, ob **virtuelle Threads oder Plattform-Threads (z.B. mit Executors.newCachedThreadPool()) verwendet werden sollen.**

JEP 436: Virtual Threads

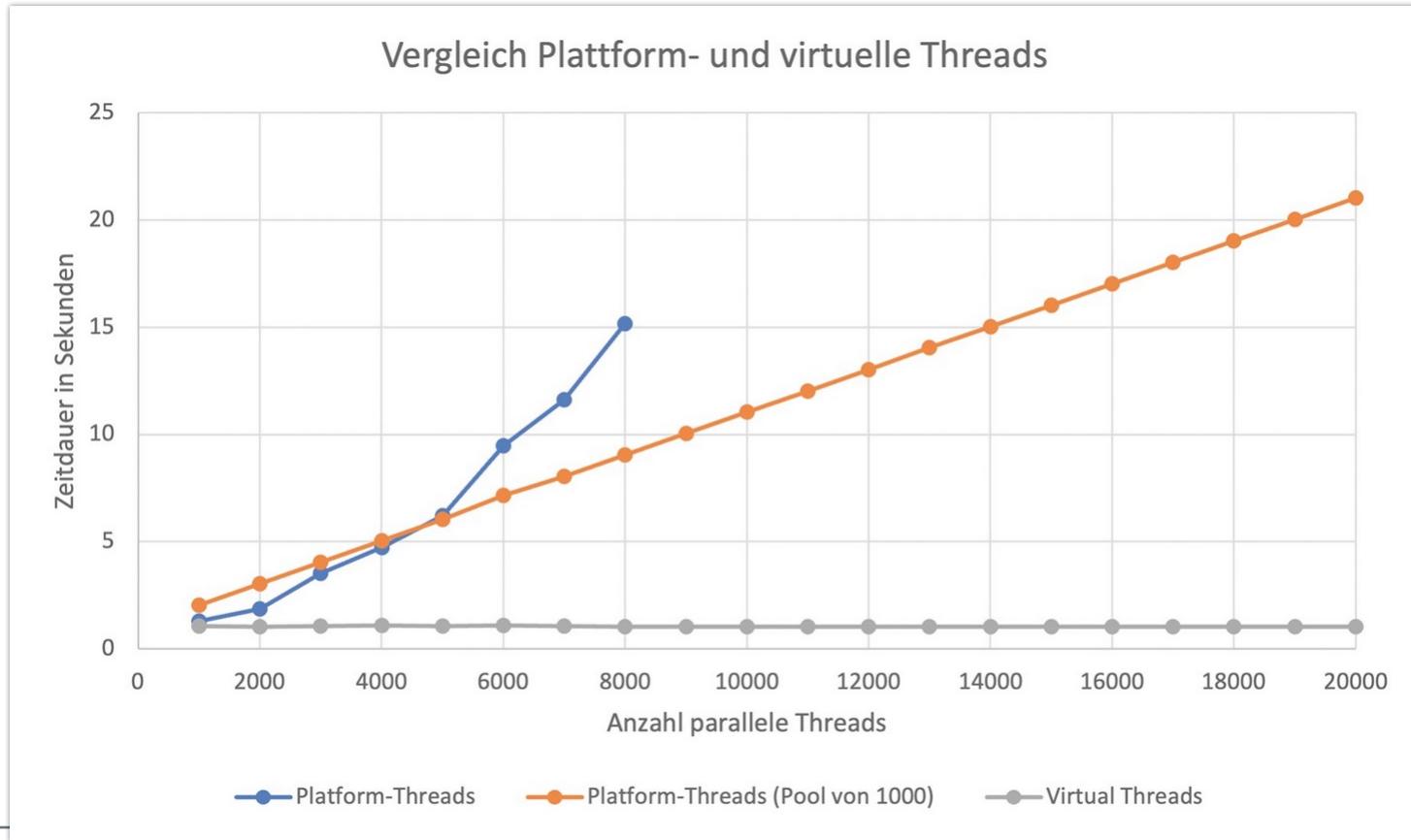


```
public static void main(String[] args)
{
    System.out.println("Start");

    try (var executor = Executors.newVirtualThreadPerTaskExecutor())
    {
        for (int i = 0; i < 10_000; i++)
        {
            final int pos = i;
            executor.submit(() -> {
                Thread.sleep(Duration.ofSeconds(1));
                return pos;
            });
        }
    }
    // executor.close() is called implicitly, and waits
    System.out.println("End");
}
```



- **Performance-Vergleich in Tausenderschritten für Plattform- und virtuelle Threads**
- **Die Threads warten jeweils eine Sekunde, um den Zugriff auf eine externe Schnittstelle zu simulieren. Am Ende wird die Gesamtzeit gemessen.**





DEMO & Hands on

PlatformThreads.java
VirtualThreads.java

```
java --enable-preview --source 20  
src/main/java/java20/virtualthreads/VirtualThreads.java
```



JEP 437: Structured Concurrency (Second Incubator in Java 20)





- Dieser JEP bringt die Structured Concurrency als Vereinfachung von Multithreading.
- Wenn eine Aufgabe aus mehreren Teilaufgaben besteht, die parallel verarbeitet werden können, ermöglicht Structured Concurrency, dies auf besonders lesbare und wartbare Weise umzusetzen.
- Betrachten wir das Ermitteln eines Users und dessen Bestellungen anhand einer User-ID:

```
static Response handleSynchronously(Long userId) throws InterruptedException {
    var user = findUser(userId);
    var orders = fetchOrders(userId);

    return new Response(user, orders)
}
```
- Beide Aktionen könnten parallel ablaufen.



- **Erster Versuch: Herkömmliche Umsetzung mit ExecutorService:**

```
static Response handleOldStyle(Long userId) throws ExecutionException,  
                                         InterruptedException  
{  
    var executorService = Executors.newCachedThreadPool();  
  
    var userFuture = executorService.submit(() -> findUser(userId));  
    var ordersFuture = executorService.submit(() -> fetchOrders(userId));  
  
    var user = userFuture.get();           // Join findUser  
    var orders = ordersFuture.get();      // Join fetchOrders  
  
    return new Response(user, orders);  
}
```

- **Wir übergeben die zwei Teilaufgaben an den Executor und warten auf die Teilergebnisse. Dieser Happy Path ist schnell implementiert.**



```
static Response handleOldStyle(Long userId) throws ExecutionException,  
                                         InterruptedException  
{  
    var executorService = Executors.newCachedThreadPool();  
  
    var userFuture = executorService.submit(() -> findUser(userId));  
    var ordersFuture = executorService.submit(() -> fetchOrders(userId));  
  
    var user = userFuture.get();           // Join findUser  
    var orders = ordersFuture.get();      // Join fetchOrders  
  
    return new Response(user, orders);  
}
```

- Weil die Teilaufgaben parallel ausgeführt werden, können sie unabhängig voneinander erfolgreich sein oder fehlschlagen. Dann kann das Handling recht kompliziert werden.
- Oftmals möchte man beispielsweise nicht, dass das zweite get() aufgerufen wird, wenn bereits bei der Abarbeitung der Methode findUser() eine Exception aufgetreten ist.



- Weil die Teilaufgaben parallel ausgeführt werden, können sie unabhängig voneinander erfolgreich sein oder fehlschlagen. Dann kann das Handling recht kompliziert werden.
- Generelle Fragestellung: Wie gehen wir mit Exceptions um?
 - Konkret, wenn in einer Teilaufgabe eine Exception auftritt, wie können wir die andere abbrechen?
 - Wie können wir die Abarbeitung der Teilaufgaben insgesamt stoppen, etwa, wenn wir die Ergebnisse nicht mehr benötigen?
- Mit einigen Tricks kann man das erreichen, der Sourcecode wird dann jedoch komplex, enthält diverse Abfragen und wird insgesamt schwierig zu verstehen und zu warten.



- Umsetzung mit Structurd Concurrency in Form der Klasse StructuredTaskScope:

```
static Response handle(Long userId) throws ExecutionException,  
                                         InterruptedException  
{  
    try (var scope = new StructuredTaskScope.ShutdownOnFailure()) {  
        var userFuture = scope.fork(() -> findUser(userId));  
        var ordersFuture = scope.fork(() -> fetchOrder(userId));  
  
        scope.join();           // Join both forks  
        scope.throwIfFailed(); // ... and propagate errors  
  
        // Here, both forks have succeeded, so compose their results  
        return new Response(userFuture.resultNow(), ordersFuture.resultNow());  
    }  
}
```

- Konkurrierende Teilaufgaben mit fork() abspalten und mit blockierendem Aufruf von join() Ergebnisse einsammeln
- join() wartet, bis alle Teilaufgaben abgearbeitet sind oder ein Fehler auftrat.



Die Klasse `StructuredTaskScope` besitzt zwei Spezialisierungen:

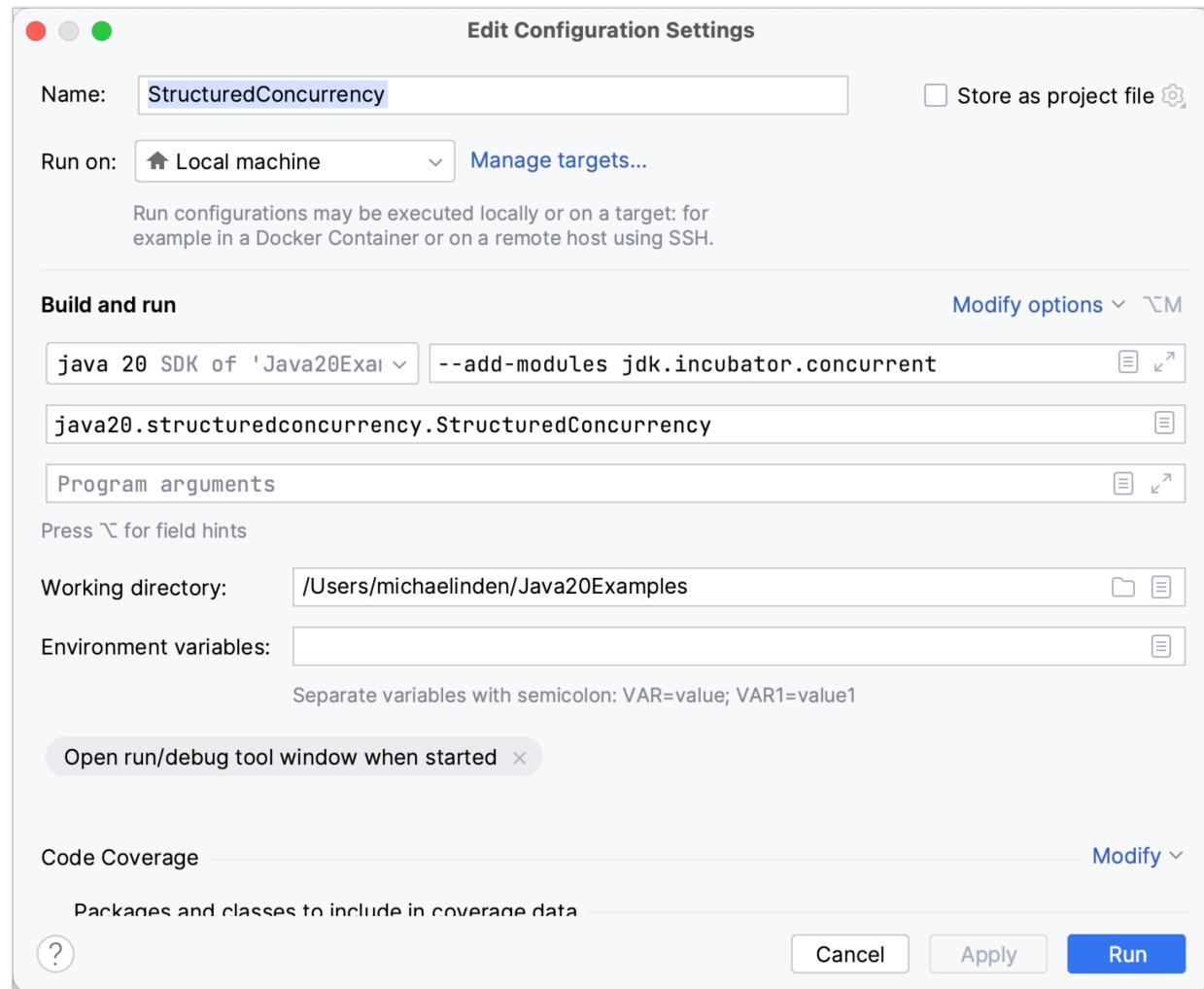
- `ShutdownOnSuccess` – ermittelt das erste eintreffende Ergebnis und beendet dann den `StructuredTaskScope`. Das hilft, wenn bereits das Ergebnis einer beliebigen Teilaufgabe ausreicht ("`invoke any`") und es nicht notwendig ist, auf die Ergebnisse anderer, nicht abgeschlossener Aufgaben zu warten.
- `Shutdown onFailure` – fängt die erste Exception ab und beendet den `StructuredTaskScope`. Diese Klasse ist dafür gedacht, wenn die Ergebnisse aller Teilaufgaben benötigt werden ("`invoke all`"); wenn eine Teilaufgabe fehlschlägt, werden die Ergebnisse der anderen, noch nicht abgeschlossenen Teilaufgaben nicht mehr benötigt.



Vorteile beim Einsatz der Klasse StructuredTaskScope:

- **Task und Subtasks bilden eine in sich geschlossene Einheiten**
- **Es werden kein ExecutorService und Threads aus einem Thread-Pool genutzt. Jede Teilaufgabe wird in einem neuen virtuellen Thread ausgeführt.**
- **Fehler in einer der Teilaufgaben => alle anderen Teilaufgaben werden abgebrochen.**
- **Wird der aufrufende Thread abgebrochen, werden auch die Teilaufgaben abgebrochen.**
- **Aufrufhierarchie (aufrufenden Thread und Teilaufgaben) im Thread-Dump gut zu erkennen.**
- Beim ExecutorService sieht man im Thread-Dump nur etwa die Thread-Namen "pool-X-thread-Y". Die Zuordnung von Pool-Thread zu aufrufenden Thread für Teilaufgaben ist kaum möglich.

JEP 437: Structured Concurrency





DEMO & Hands on

StructuredConcurrency.java

```
java --enable-preview --source 20 --add-modules jdk.incubator.concurrent  
src/main/java/java20/structuredconcurrency/StructuredConcurrency.java
```

JEP 437: Structured Concurrency und JShell



```
jshell --enable-preview --add-modules jdk.incubator.concurrent
| Willkommen bei JShell – Version 20
| Geben Sie für eine Einführung Folgendes ein: /help intro
```

```
jshell> import java.time.Duration;
...> import java.util.List;
...> import java.util.concurrent.ExecutionException;
...> import java.util.concurrent.Executors;
...>
...> import jdk.incubator.concurrent.StructuredTaskScope;
```

```
jshell> record Order(String orderNo, List<String> items) {}
| Erstellt Datensatz Order
```

```
jshell> record Response(String user, List<Order> orders) { }
| Erstellt Datensatz Response
```

JEP 437: Structured Concurrency und JShell



```
jshell> static String findUser(Long userId) throws InterruptedException {  
...>     Thread.sleep(Duration.ofSeconds(1));  
...>     return "Michael";  
...> }  
...>  
...> static List<Order> fetchOrders(Long userId) throws InterruptedException  
{  
...>     Thread.sleep(Duration.ofSeconds(1));  
...>     return List.of();  
...> }  
| Erstellt Methode findUser(Long)  
| Erstellt Methode fetchOrders(Long)
```

JEP 437: Structured Concurrency und JShell



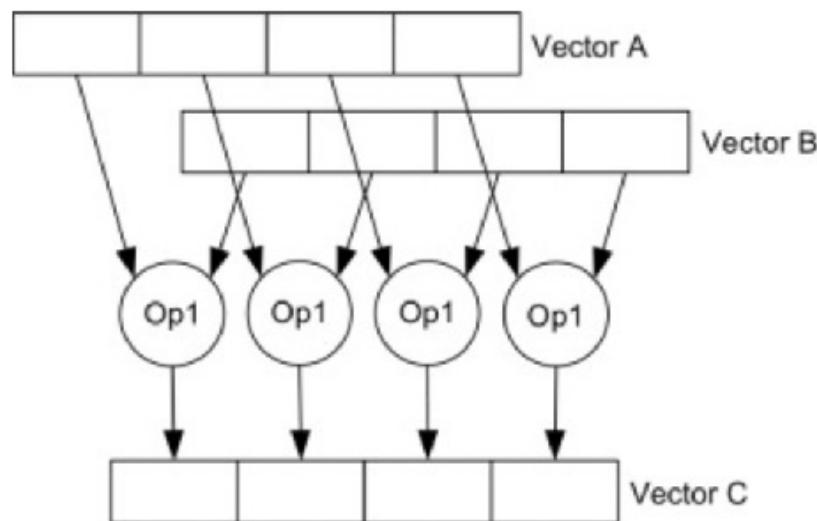
```
jshell> static Response handle(Long userId) throws ExecutionException,  
InterruptedException  
...> {  
...>     try (var scope = new StructuredTaskScope.ShutdownOnFailure()) {  
...>         var userFuture = scope.fork(() -> findUser(userId));  
...>         var ordersFuture = scope.fork(() -> fetchOrders(userId));  
...>  
...>         scope.join();           // Join both forks  
...>         scope.throwIfFailed(); // ... and propagate errors  
...>  
...>         // Here, both forks have succeeded, so compose their results  
...>         return new Response(userFuture.resultNow(),  
...>                               ordersFuture.resultNow());  
...>     }  
...> }  
| Erstellt Methode handle(Long)
```

```
jshell> handle(4711L)  
$15 ==> Response[user=Michael, orders=[]]
```



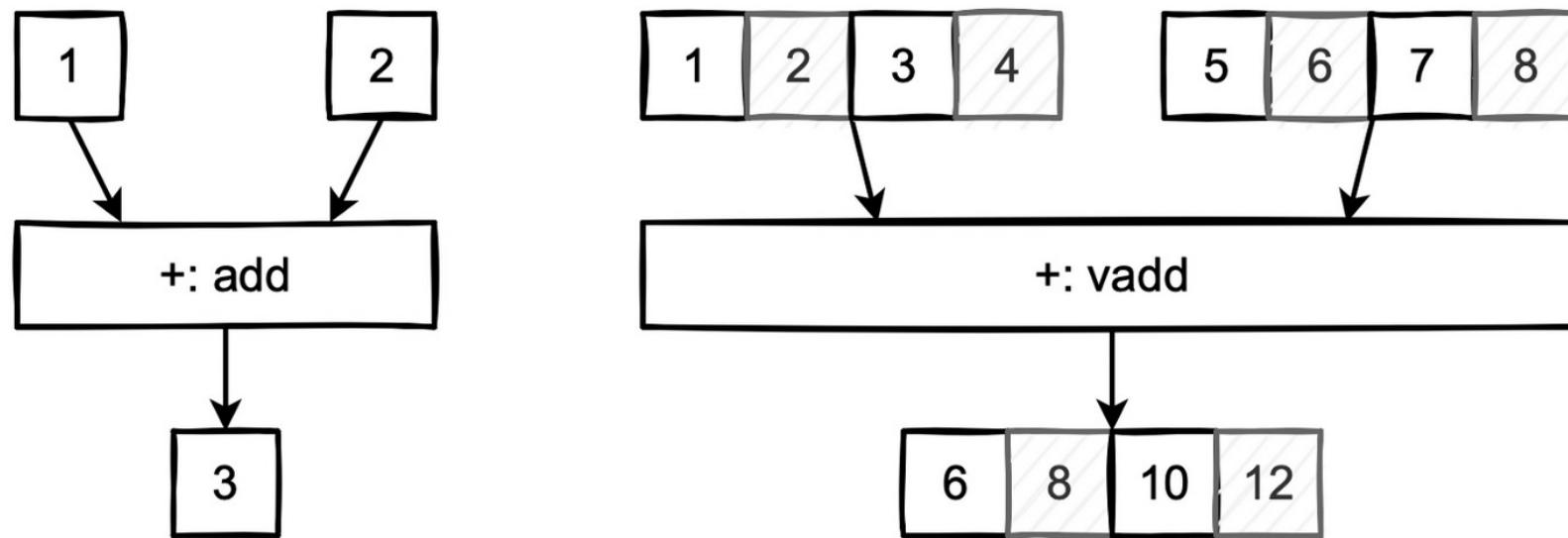
JEP 438: Vector API

(Fifth Incubator in Java 20)





- Dieser JEP hat nichts mit der Klasse `java.util.Vector` zu tun! Vielmehr geht es um eine plattformunabhängige Unterstützung von sogenannten Vektorberechnungen.
- Moderne Prozessoren können beispielsweise eine Addition oder Multiplikation nicht nur für zwei Werte, sondern für eine Vielzahl von Werten ausführen. Man spricht dabei auch von **Single Instruction Multiple Data (SIMD)**. Die folgende Grafik visualisiert das Grundprinzip:





- Das Vector API zielt darauf ab, die Leistung von Vektorberechnungen zu verbessern.
- Eine Vektorberechnung besteht aus einer Folge von Operationen mit Vektoren. Dabei kann man sich einen Vektor wie ein Array von primitiven Werten vorstellen.
- Wollte man nun zwei Vektoren respektive Arrays mit einer mathematischen Operation verknüpfen, so würde man dazu herkömmlich eine Schleife über alle Werte nutzen.

```
int[] a = { 1, 2, 3, 4, 5, 6, 7, 8 };
int[] b = { 1, 2, 3, 4, 5, 6, 7, 8 };

var c = new int[a.length];
for (int i = 0; i < a.length; i++)
{
    c[i] = a[i] + b[i];
}
```



- Mit dem **Vector API** lassen sich die Besonderheiten und Optimierungen in modernen Prozessoren ausnutzen. Dazu gibt man gewisse Hilfestellungen für die Berechnungen vor.

```
int[] a = { 1, 2, 3, 4, 5, 6, 7, 8 };
int[] b = { 1, 2, 3, 4, 5, 6, 7, 8 };
```

```
var c = new int[a.length];
var vectorA = IntVector.fromArray(IntVector.SPECIES_256, a, 0);
var vectorB = IntVector.fromArray(IntVector.SPECIES_256, b, 0);
var vectorC = vectorA.add(vectorB);
// var vectorC = vectorA.mul(vectorB);
vectorC.intoArray(c, 0);
```

- Das steuernde Element ist hier die Größe des Vektors, den wir durch das Argument `IntVector.SPECIES_256` auf 256 Bit festlegen.
- Danach kann dann die passende Aktion erfolgen, hier `add()` oder `mul()`.



- Bei nicht perfekt passenden und größeren Ausgangsdaten müssen die Aktionen passend aufgeteilt werden.
- Betrachten wir das Beispiel aus JEP 417 und die Skalarberechnung als Ausgangsbasis:

```
void scalarComputation(float[] a, float[] b, float[] c)
{
    for (int i = 0; i < a.length; i++)
    {
        c[i] = (a[i] * a[i] + b[i] * b[i]) * -1.0f;
    }
}
```

- Algorithmus ist absolut verständlich und leicht nachvollziehbar

JEP 438: Vector API



```
static final VectorSpecies<Float> SPECIES = FloatVector.SPECIES_PREFERRED;

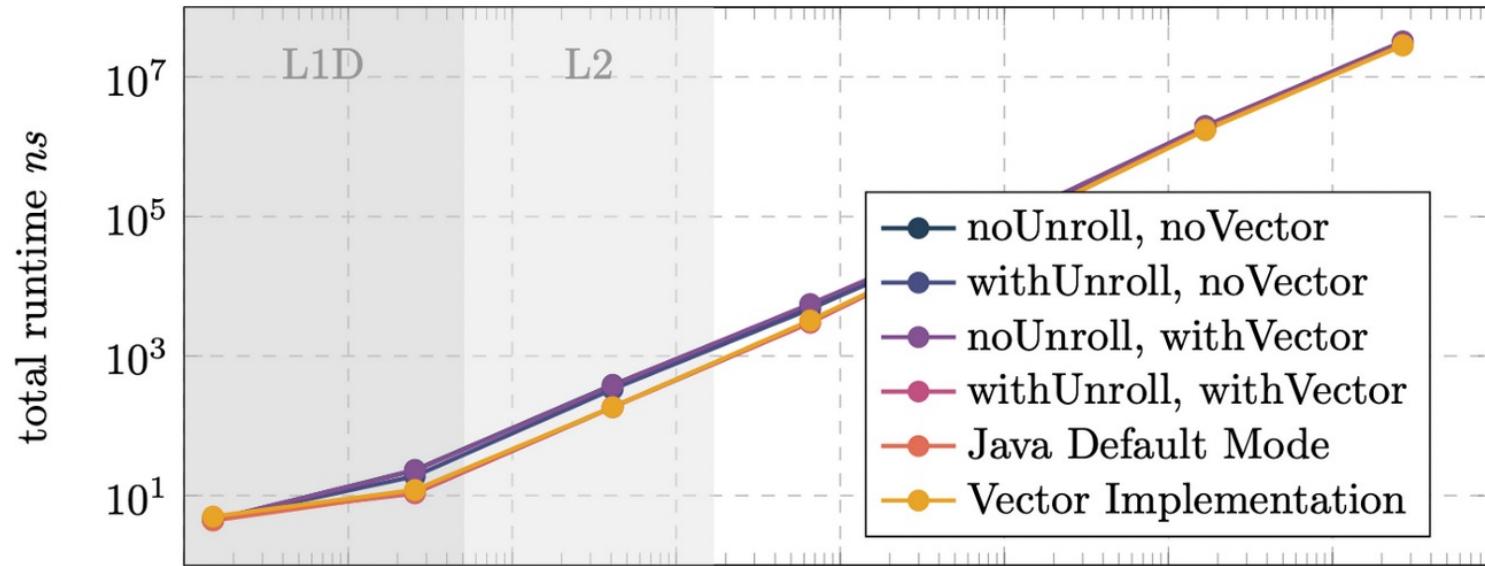
void vectorComputation(float[] a, float[] b, float[] c)
{
    int i = 0;
    int upperBound = SPECIES.loopBound(a.length);
    for (; i < upperBound; i += SPECIES.length())
    {
        var va = FloatVector.fromArray(SPECIES, a, i);
        var vb = FloatVector.fromArray(SPECIES, b, i);
        var vc = va.mul(va)
                  .add(vb.mul(vb))
                  .neg();
        vc.intoArray(c, i);
    }

    for (; i < a.length; i++)
    {
        c[i] = (a[i] * a[i] + b[i] * b[i]) * -1.0f;
    }
}
```

JEP 438: Vector API Benchmarking

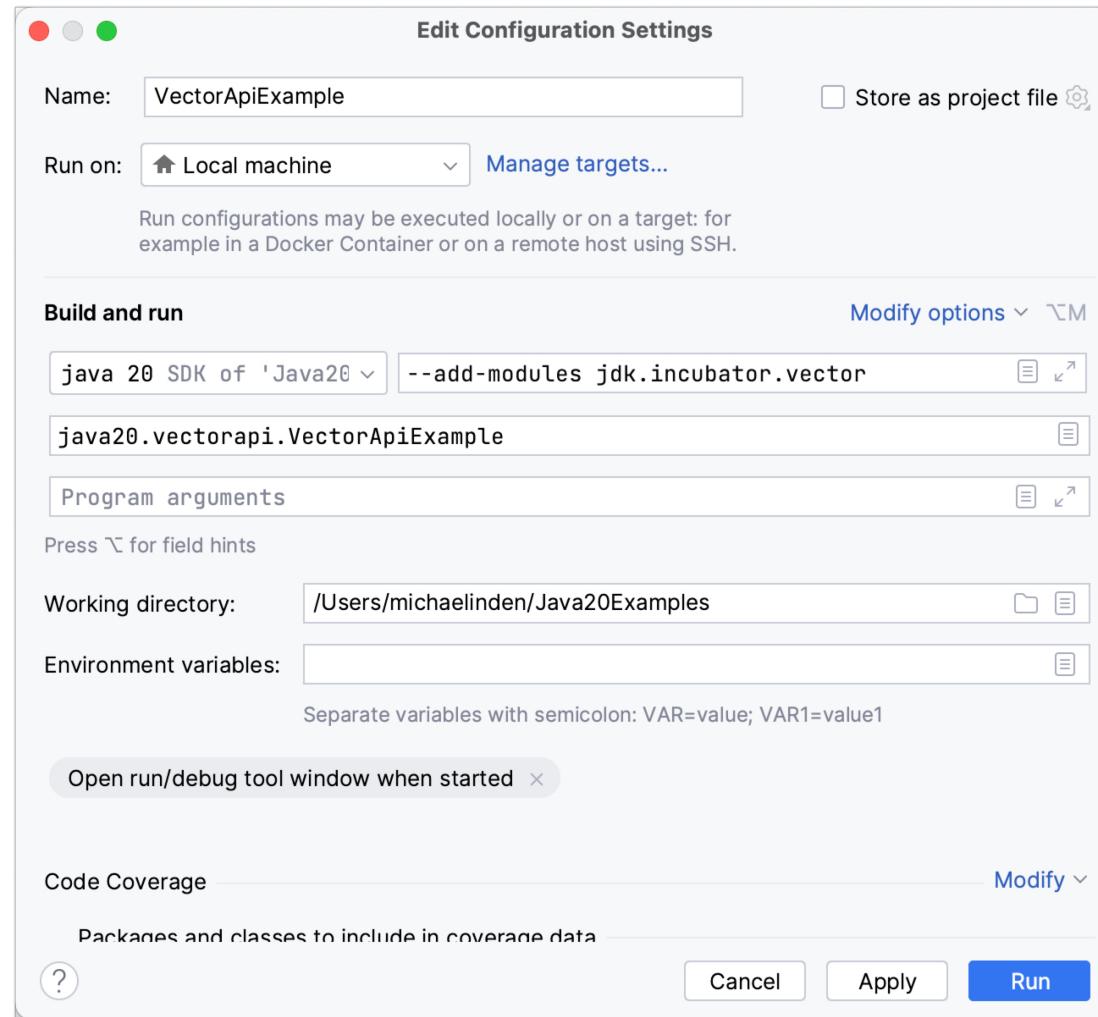


Benchmark: $c[n] = a[n] + b[n]$



Method	Peak Speed-Up
No Unroll, No Vector	1.00x
With Unroll, No Vector	1.22x
No Unroll, With Vector	1.01x
With Unroll, With Vector	2.15x
Java Default	2.06x
Vector Implementation	2.08x

JEP 438: Vector API



JEP 438: Vector API (in Konsole und JShell)



```
$ java --enable-preview --source 20 --add-modules jdk.incubator.vector  
src/main/java/java20/vectorapi/VectorApiExample.java
```

```
WARNING: Using incubator modules: jdk.incubator.vector  
[2, 4, 6, 8, 10, 12, 14, 16]
```

```
$ jshell --enable-preview --add-modules jdk.incubator.vector  
| Willkommen bei JShell – Version 20  
| Geben Sie für eine Einführung Folgendes ein: /help intro  
  
jshell> import jdk.incubator.vector.FloatVector;  
  
jshell> import jdk.incubator.vector.IntVector;  
  
jshell> import jdk.incubator.vector.VectorSpecies;
```



Misc





- **java.net.URL Konstruktoren sind deprecated**

```
// old style
URL url = new URL("https://www.happycoders.eu"); // deprecated
```

```
// new style
URL url2 = URI.create("https://www.happycoders.eu").toURL();
```

- **java.util.Locale Konstruktoren sind deprecated (bereits in Java 19)**

```
Locale german1 = new Locale("de"); // deprecated
Locale germany1 = new Locale("de", "DE"); // deprecated
```

```
Locale german2 = Locale.of("de");
Locale germany2 = Locale.of("de", "DE");
```

Größenvorgaben von Collections



- **java.util.List erwartungskonform**

```
List<String> presizedList = new ArrayList<>(2000);
```

- **java.util.HashMap erwartungskonform, worin besteht der Unterschied???**

```
Map<String, String> presizedHashMap = new HashMap<>(2000);
```

```
Map<String, String> presizedHashMap2 = HashMap.<String, String>newHashMap(2000);
```



Übungen PART 6

https://github.com/Michaeli71/Governikus_Java_Update_11_17



Ausblick Java 21

Features

- 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)

String Templates



- **String-Prozessor STR in Kombination mit Platzhaltern mit \{varName}**
- **Beispiel**

```
String old = "The file " + filePath + " " + (file.exists() ? "does" : "does not" + " exist");
```

```
String msg = STR."The file \{filePath\} \{file.exists() ? "does" : "does not"\} exist";
```

- **Beispiel 2**

```
String firstName = "Michael";
String lastName = "Inden";
String firstLastName = STR."\{firstName\} \{lastName\}";
String lastFirstName = STR."\{lastName\}, \{firstName\}";
System.out.println(firstLastName);
System.out.println(lastFirstName);
```

- => Michael Inden
Inden, Michael

String Templates – Berechnungen



```
int x = 10, y = 20;  
String calculation = STR."\{x} + \{y} = \{x + y}";  
System.out.println(calculation);
```

- => 10 + 20 = 30

```
int index = 0;  
String modifiedIndex = STR."\{index++}, \{index++}, \{index++}, \{index++}";  
System.out.println(modifiedIndex);
```

- => 0, 1, 2, 3

```
String currentTime = STR."The time is \{  
    DateTimeFormatter.ofPattern("HH:mm:ss").format(LocalTime.now())\} right now";  
System.out.println(currentTime);
```

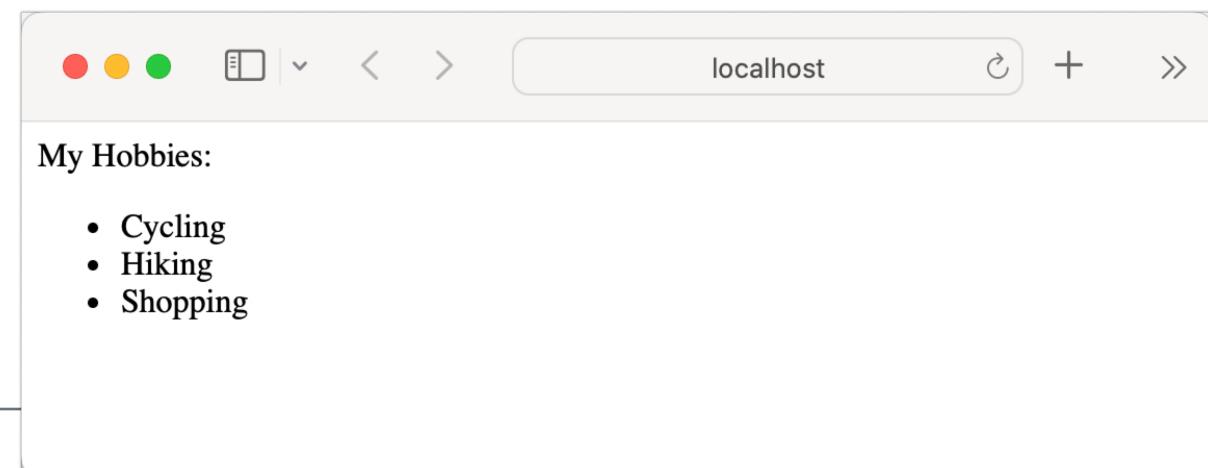
- => The time is 11:54:35 right now

String Templates



```
String title = "My First Web Page";
String text = "My Hobbies:";
var hobbies = List.of("Cycling", "Hiking", "Shopping");
String html = STR. """
<html>
  <head><title>\{title}</title></head>
  <body>
    <p>\{text}</p>
    <ul>
      <li>\{hobbies.get(0)}</li>
      <li>\{hobbies.get(1)}</li>
      <li>\{hobbies.get(2)}</li>
    </ul>
  </body>
</html>""";
System.out.println(html);
```

```
<html>
  <head><title>My First Web Page</title></head>
  <body>
    <p>My Hobbies:</p>
    <ul>
      <li>Cycling</li>
      <li>Hiking</li>
      <li>Shopping</li>
    </ul>
  </body>
</html>
```



Sequenced Collection



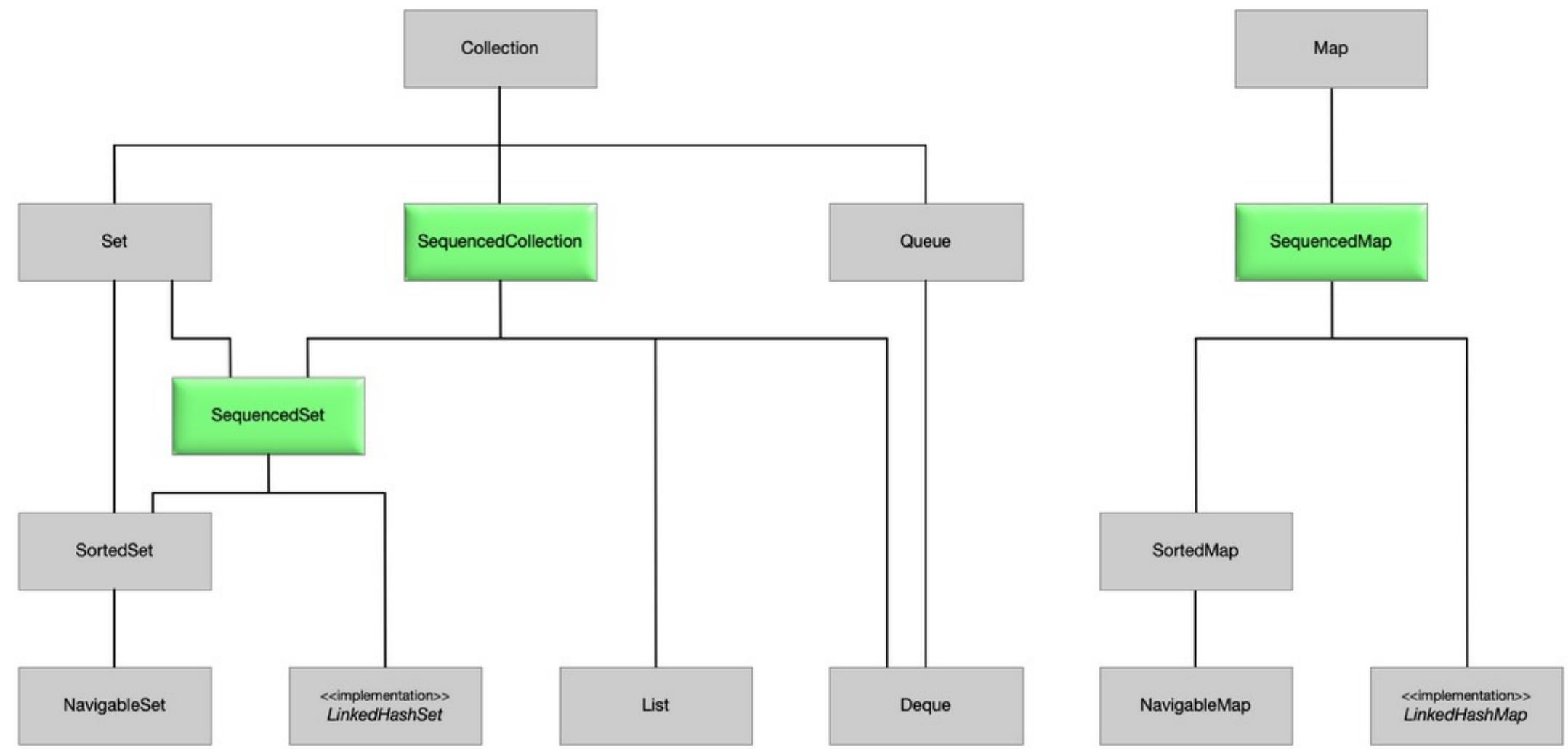
- **Früher**

First element	Last element
List list.get(0)	list.get(list.size() - 1)
Deque deque.getFirst()	deque.getLast()
SortedSet sortedSet.first()	sortedSet.last()
LinkedHashSet linkedHashSet.iterator().next() // missing	

- **Zukunft**

```
interface SequencedCollection<E> extends Collection<E> {  
    // new method  
    SequencedCollection<E> reversed();  
    // methods promoted from Deque  
    void addFirst(E);  
    void addLast(E);  
    E getFirst();  
    E getLast();  
    E removeFirst();  
    E removeLast();  
}
```

Sequenced Collection



Sequenced Collection



```
public class SequencedCollectionExample {  
    public static void main(String[] args) {  
        SequencedCollection<String> sc = List.of("A", "B", "C");  
        System.out.println(sc.getFirst() + " / " + sc.getLast());  
        sc.reversed().forEach(System.out::println);  
  
        System.out.println(Set.of("A", "B", "C", "D"));  
        SequencedCollection<String> sc2 = new TreeSet<>((Set.of("A", "B", "C", "D")));  
        System.out.println(sc2.getFirst() + " / " + sc2.getLast());  
    }  
}
```

- =>

A / C
C
B
A
[C, B, A, D]
A / D

Unnamed classes and instance main() method



- **Früher**

```
public class InstanceMainMethodOld {  
    public static void main(final String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```

- **Kürzer**

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

- **Noch kürzer:**

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



Fazit



Positives



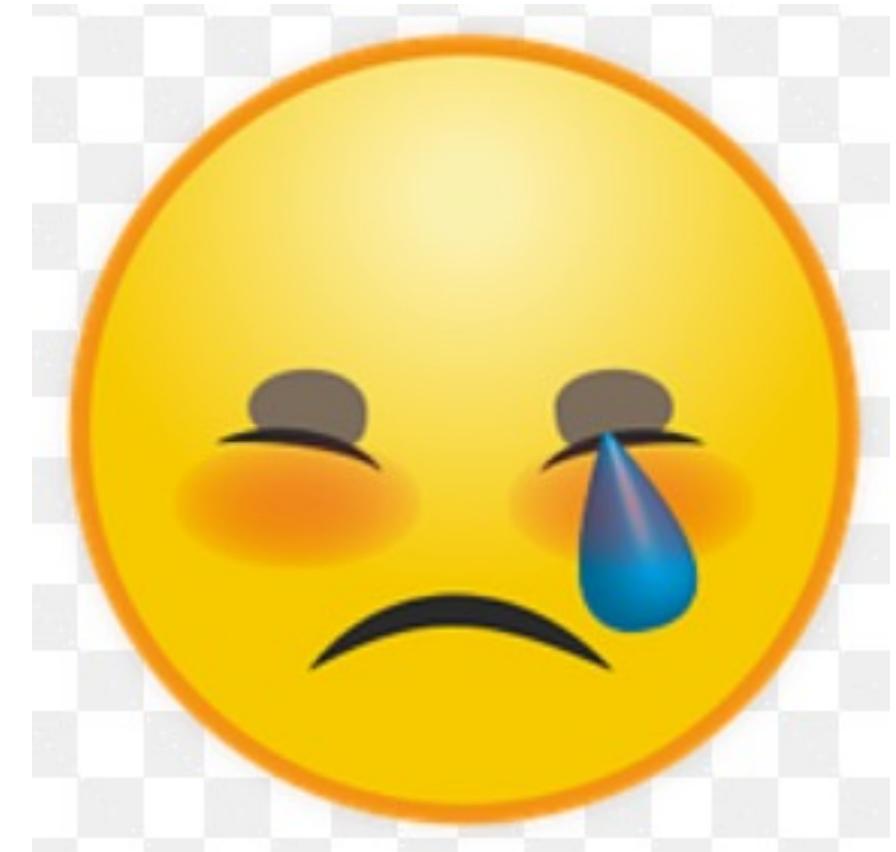
- eine paar Dinge aus Project COIN und in der Syntax
- Switch / Records
- diverse praktische Erweiterungen in den APIs
- JPackage
- HTTP 2 (Java 11)
- Modularisierung mit Project JIGSAW (Java 9) – aber leider (immer noch) kein wirklich gutes Tooling

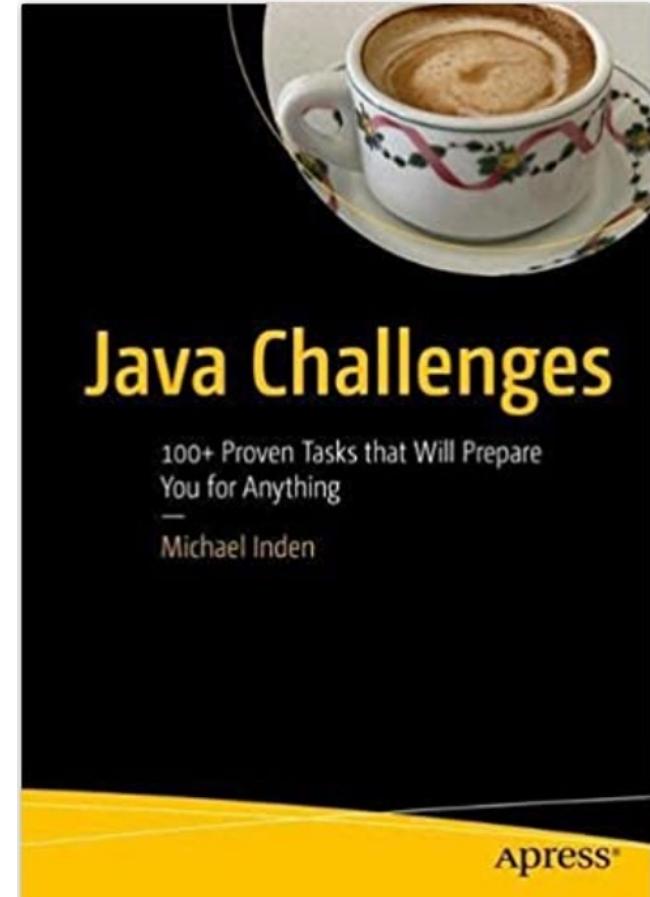


Negatives



- Folge-Release waren zwar pünktlich, aber manchmal (außer Java 14) ziemlich dünn bezüglich wichtigen Neuerungen
- Immer Mal wieder weniger als geplant
 - keine Versionierung bei JIGSAW
 - kein JSON-Support
 - statt Collection-Literals nur Convenience-Methods
 - Statt ZIP nur TEE (ing)-Kollektor







Questions?



Thank You