GraalVM: Neue universelle Maschine S. 10
Grundlagen: Koroutinen in Kotlin S. 19
Dokumentieren von REST APIs S. 42

JAVA Mag

Java magazin™

Java | Architektur | Software-Innovation

JAVA 12

www.javamagazin.de

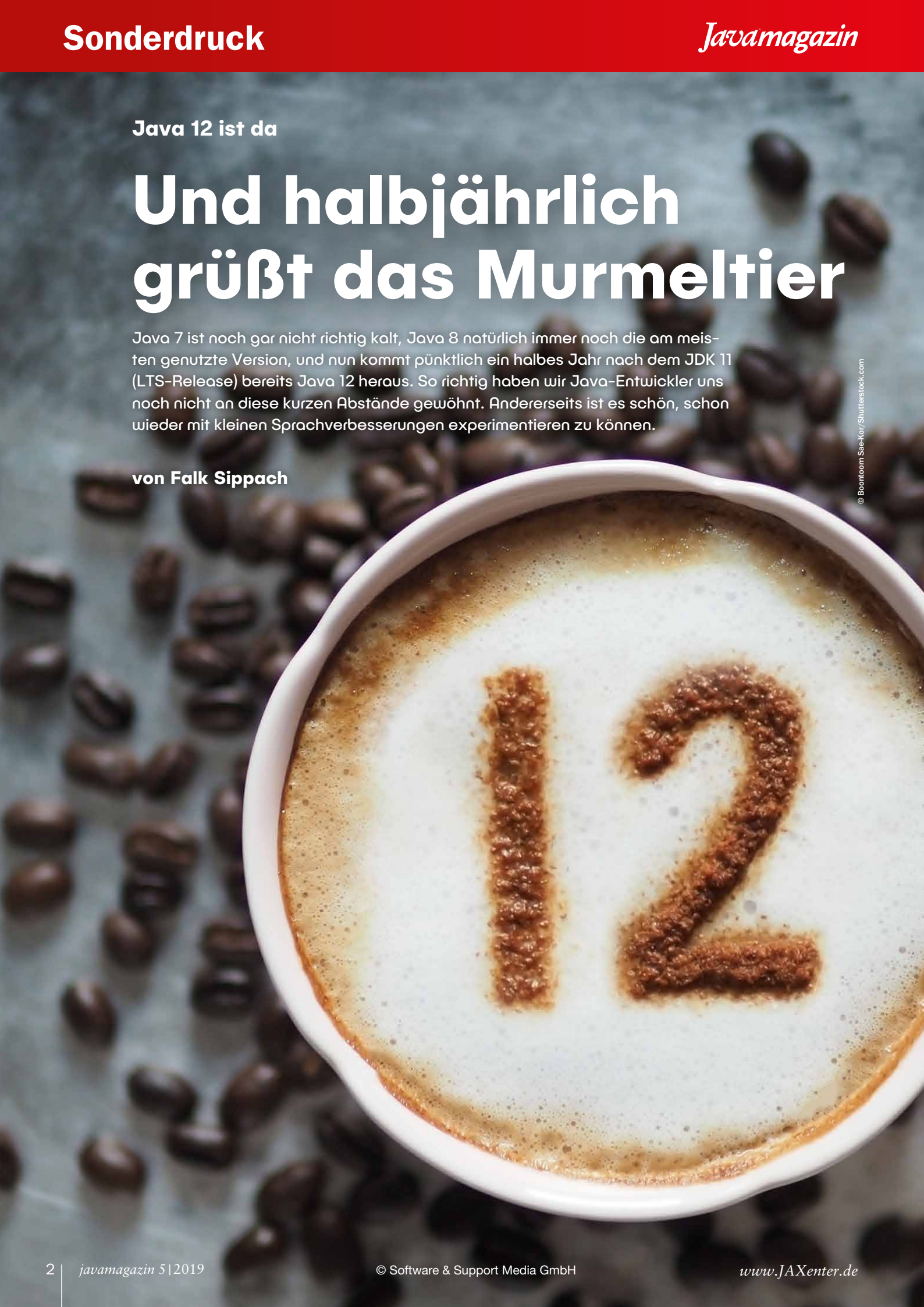
Ausgabe 5.2019

Java 12 ist da

Und halbjährlich grüßt das Murmeltier

Java 7 ist noch gar nicht richtig kalt, Java 8 natürlich immer noch die am meisten genutzte Version, und nun kommt pünktlich ein halbes Jahr nach dem JDK 11 (LTS-Release) bereits Java 12 heraus. So richtig haben wir Java-Entwickler uns noch nicht an diese kurzen Abstände gewöhnt. Andererseits ist es schön, schon wieder mit kleinen Sprachverbesserungen experimentieren zu können.

von Falk Sippach



In das JDK 12 haben es die folgenden Java Enhancement Proposals (JEP) geschafft [1]:

- JEP 189: Shenandoah: A Low-Pause-Time Garbage Collector (Experimental)
- JEP 230: Microbenchmark Suite
- JEP 325: Switch Expressions (Preview)
- JEP 334: JVM Constants API
- JEP 340: One AArch64 Port, Not Two
- JEP 341: Default CDS Archives
- JEP 344: Abortable Mixed Collections for G1
- JEP 346: Promptly Return Unused Committed Memory from G1

Auf den ersten Blick sieht es nicht so viel aus, insbesondere, wenn man die langen Änderungslisten der letzten großen Releases im Hinterkopf hat, wie zum Beispiel die Lambdas und das Stream API in Java 8. Oder wenn man sich möglicherweise die Neuerungen im Fall des JDK 9 (Plattformmodulsystem, JShell ...) noch gar nicht näher angeschaut, geschweige denn sie verdaut hat.

Die Idee der halbjährlichen Releasezyklen ist es, dass neue Features parallel und ohne Zeitdruck unabhängig von den nächsten Veröffentlichungsterminen entwickelt werden können. Die Aufnahme in das entsprechende Release erfolgt erst, wenn die Funktion ausgereift ist. Dadurch sind die Änderungen der einzelnen Releases natürlich überschaubar. Andererseits ist schon lange im Voraus klar, wann die nächste Version erscheinen wird; sie kann sich nicht mehr auf unbestimmte Zeit verzögern.

Switch Expressions

Wenn man die obige Featureliste durchschaut, sticht aus Entwicklersicht hauptsächlich ein Punkt ins Auge: Switch Expressions. Nun ist das Konstrukt Switch eigentlich eine Idee aus der prozeduralen Programmierung und ein Überbleibsel aus der Tatsache, dass sich die Syntax von Java initial stark an C orientiert hat. Für den erfahrenen Programmierer ist das Switch Statement aber ein Code Smell und kann im Sinne der Objektorientierung besser mit dem Strategieentwurfsmuster ausgedrückt werden. Nichtsdestotrotz gibt es Situationen, in denen man mit einem Switch sehr effizient und übersichtlich Fallunterscheidungen umsetzen kann. Allerdings hatte die Implementierung bisher einige Schwächen. So konnte man anfänglich nur auf Zahlen prüfen, ab Java 5 dann zwar auch auf Enums (verkappte Integer) und seit Java 7 immerhin auf Strings. In vielen Programmiersprachen gibt es dagegen keine Einschränkungen über die verwendbaren Datentypen. Zudem ist das in funktionalen Sprachen verbreitete Pattern Matching eine Art „Switch on Steroids“. Mit dem JEP 305 gibt es sogar Pläne, Pattern Matching in zukünftigen Java-Versionen einzuführen. Die Switch Expressions kann man schon mal als eine Art Vorgeschmack darauf verstehen. Nicht vergessen sollten wir aber, dass es sich aktuell im JDK 12 nur um eine Preview handelt. Bis zum nächsten LTS-Release (JDK 17) kann sich durch Feedback aus der Community an der Umsetzung noch einmal einiges ändern oder möglicherweise auch wieder ganz verschwinden. Werfen wir in Listing 1 zunächst einen Blick auf das klassische Switch Statement.

Probleme mit dem Switch Statement

Diese Variante wirkt etwas unhandlich und wird bei komplexeren *case*-Blöcken schnell unübersichtlich. Zudem können kleine Flüchtigkeitsfehler oder Unwissenheit über die genaue Funktionsweise zu schlecht nachvollziehbaren Bugs führen. Ein möglicherweise unabsichtlicher Fall Through entsteht, wenn man einen *case*-Block nicht mit einem *break* beendet. So springt die Ausführung weiter in den nächsten Block, auch wenn dessen Bedingung gar nicht zur Switch-Variable passt. In unserem Beispiel würde bei der Ausführung mit *i = 1* als Ergebnis *two* zurückgegeben. Sinnvoll kann dieses Verhalten wiederum sein, wenn mehrere Switch-Bedingungen auf das gleiche Ergebnis abgebildet werden sollen, wobei dann die durchfallenden *case*-Zweige explizit leer bleiben (Listing 2).

Sollte man vergessen, einen bestimmten Fall abzudecken, wird der Switch scheinbar nicht ausgeführt. Daher empfiehlt es sich immer, einen Defaultzweig zu definieren.

Listing 1

```
public String describeInt(int i) {
  String str = "not set";
  switch(i) {
    case 1:
      str = "one";
    case 2:
      str = "two";
      break;
    case 3:
      str = "three";
      break;
  }
  return str;
}
```

Listing 2

```
public String describeInt(int i) {
  String str = "not set";
  switch(i) {
    case 1:
    case 2:
      str = "one or two";
      break;
    case 3:
      str = "three";
      break;
  }
  return str;
}
```

Dieser kann dann wenigstens zur Laufzeit einen Fehler für nichtdefinierte Inputparameter werfen. Denn leider warnt uns der Compiler nicht, wenn wir den Switch mit Werten kleiner als 1 oder größer als 3 aufrufen.

In den unterschiedlichen *case*-Blöcken wird typischerweise redundant immer wieder die gleiche temporäre Variable mit natürlich unterschiedlichen Werten befüllt. Das widerspricht dem DRY-Prinzip und erschwert die Nachvollziehbarkeit, insbesondere wenn mit dem Fall Through gearbeitet wird. Temporäre Variablen sind nicht umsonst ein Code Smell. Idealerweise kann man komplett auf sie verzichten.

Zu guter Letzt führt der globale Scope des Switch Statements dazu, dass in den einzelnen Blöcken möglicherweise unterschiedliche Variablen definiert werden müssen, um ungewollte Nebeneffekte bei der Verwendung der gleichen Variablennamen zu vermeiden.

Switch in Java 12

Werfen wir nun einen Blick auf die Neuerungen der Preview in Java 12. So kann ein *case*-Block jetzt mehrere Labels haben und ermöglicht damit eine redundanzfreie und kompaktere Schreibweise, wenn wir einen Fall Through verwenden (Listing 3).

Mit der Pfeilsyntax existiert des Weiteren eine Variante, die der Definition von Lambdaausdrücken ähnelt. Jeder Zweig wird damit auf eine Zeile reduziert und das *break* entfällt. Das spart wieder Boiler-Plate-Code und macht Switch Statements kompakter und übersichtlicher. Ein unabsichtlicher Fall Through ist nicht mehr möglich. Es wird immer nur genau das eine Statement auf der rechten Seite des Pfeils ausgeführt. Sobald mehr als ein Statement auf der rechten Seite benötigt wird, muss der Code innerhalb eines Blocks stehen. Damit wird ein eigener Scope geschaffen, der eine kollisionsfreie Variablendeklaration ermöglicht (Listing 4).

In der funktionalen Programmierung gibt es typischerweise keine Zustandsänderungen in Form von Variablenzuweisungen. Vielmehr wird mit Ausdrücken gearbeitet, die wiederum als Inputparameter einer anderen Funktion zur weiteren Bearbeitung übergeben werden können. Die Switch Expression ist jetzt auch ein Ausdruck, der ein Ergebnis zurückliefert. So kann man das Resultat zum Beispiel direkt einer Variablen (besser Konstanten) zuweisen oder einem Methodenaufruf als Parameter über- bzw. per *return* zurückgeben (Listing 5).

Jeder Zweig muss mit dem *break* jeweils genau einen Wert zurückliefern („Break with Value"-Semantik). Dadurch kann der Compiler prüfen, ob wir alle möglichen Fälle behandeln oder wenigstens einen Defaultblock als Fallback angegeben haben. Damit ist sichergestellt, dass immer genau ein Zweig abgearbeitet und dessen Ergebnis zurückgeliefert wird. Switch Statements ohne sichtbare Ergebnisse gehören somit der Vergangenheit an.

Durch die schon angesprochene alternative Pfeilnotation verkürzt sich die Syntax der Switch Expression nochmals (Listing 6).

Die Switch Expression arbeitet natürlich auch nahtlos mit der in Java 10 eingeführten Local Variable Type Inference zusammen. Bei der Zuweisung zu einer mit *var* deklarierten Variable wird der spezifischste gemeinsame Typ aller Case-Zweige (kleinster gemeinsamer Nenner) ausgewählt.

Ein Nachteil von Switch Expressions ist, dass eine Expression letztendlich immer zu einem Wert aufgelöst werden muss. Dadurch ist es nicht möglich, innerhalb

Listing 3

```
public String describeInt(int i) {
  String str = "not set";
  switch (i) {
    case 1, 2:
      str = "one or two";
      break;
    case 3:
      str = "three";
      break;
  }
  return str;
}
```

Listing 4

```
public String describeInt(int i) {
  String str = "not set";
  switch (i) {
    case 1, 2 -> str = "one or two";
    case 3 -> {
      var i = complexComputation();
      str = "three" + i;
    }
  }
  return str;
}
```

Listing 5

```
public static String describeInt(int i) {
  return switch (i) {
    case 1, 2:
      break "one or two";
    case 3:
      break "three";
    default:
      break "smaller than one or bigger than three";
  };
}
```

der Expression *return* oder *continue* zu benutzen. Das wurde bisher häufig verwendet, um direkt aus der Methode bzw. Iteration zu springen. Es ist allerdings weiterhin möglich, eine Exception zu werfen und damit die Expression frühzeitig zu beenden.

Bisher kann man in einem Switch die Datentypen *byte*, *short*, *char*, *int* und deren Wrapper-Klassen sowie Enums und Strings verwenden. Es gibt bereits Pläne, zusätzlich *float*, *double* und *long* zu unterstützen. Mit dem ebenfalls geplanten Pattern Matching kann man letztlich über beliebige Datentypen switchen. Die Idee des Pattern Matchings ist aber eine ganz andere, es wird zur Destrukturierung von Objekten eingesetzt.

Um die Switch Expressions bereits vor dem offiziellen Erscheinungstermin von Java 12 ausprobieren zu können, hatten manche IDEs (z. B. IntelliJ IDEA 2019.1) schon frühzeitig Support für das neue Previewfeature angeboten. Um es mit einer Early-Access-Version des JDK 12 testen zu können, musste man beim Starten des Java Interpreters zusätzlich die Parameter *--enable-preview* und *--source 12* mitgeben: *java --enable-preview --source 12 SwitchExample.java*.

Ausblick Raw String Literals

Neben der Switch Expression gab es noch diverse weitere Neuerungen. So wird ab dem JDK 12 eine Microbenchmark Suite ausgeliefert, die auf dem bekannten Java Microbenchmark Harness (JMH) basiert. Damit kann man nun mit JDK-Bordmitteln sehr einfach Benchmarks erstellen. Das JVM Constants API ermöglicht das typsichere und damit weniger fehleranfällige Laden von Werten aus dem Java Constants Pool (*int*, *float*, *String*, *Class*), ist aber eher ein JDK-internes Feature und wird vermutlich nur selten von einem Java-Programmierer benötigt werden.

Bei den Garbage Collectors gibt es ebenfalls einige Neuerungen. So hat der Default-GC G1 Performanceverbesserungen erfahren. Mit Shenandoah wurde zudem ein neuer, noch experimenteller Garbage Collector eingeführt. Er wurde ursprünglich von Red Hat entwickelt und ist eine Alternative zum bereits existierenden ZGC [2], verwendet aber einen anderen Algorithmus. Dabei sind die STW-(Stop-the-World-)Pausen sehr kurz, und es können große Mengen an Hauptspeicher (mehrere Terabyte Heap) effizient verwaltet werden. Dazu werden die Aufräumarbeiten konkurrierend zur eigentlichen Anwendung erledigt, was diese dann allerdings etwas langsamer macht. Um beim Starten von Java-Anwendungen nicht immer alle Metadaten der Klassen erneut laden zu müssen, werden diese Informationen in sogenannten Class-Data-Sharing-Archiven abgelegt, was die Startzeit verkürzt, die Performance der Garbage Collection verbessert und den Memory Footprint verringert.

Auch in der JDK-Klassenbibliothek gibt es ein paar Änderungen [3]. So wurden de Klasse *String* zwei neue Methoden spendiert, um entweder einen Text einzurücken (*indent*) oder umzuwandeln (*transform*). Und die Methode *mismatch* in *Files* findet das erste Byte, das sich zwischen zwei Dateien unterscheidet (Listing 7).

Ausblick Raw String Literals

Ursprünglich sollten neben den Switch Expressions auch die Raw String Literals in Java 12 erscheinen. Allerdings wurde die Veröffentlichung dieses Features auf unbestimmte Zeit verschoben, weil noch der nötige Feinschliff gefehlt hat. Werfen wir aber bereits jetzt einen kurzen Blick darauf, was uns in einem der nächsten Releases erwarten wird. Die Hauptprobleme in den klassischen Java-Strings sind der unintuitive Einsatz von Zeilenumbrüchen (*1\n2\n3*) und das schlecht lesbare Escapen des Backslashs, was insbesondere Windows-Dateipfade und reguläre Ausdrücke unnötig verkompliziert (*c:\\tmp bzw. \\d{2}\\.\\d{3}*).

Raw-String-Literale interpretieren abgesehen von CRLF bzw. LF keinerlei Escape-Sequenzen. Als Trennzeichen fungiert der Backtick:

```
var string = `1. Zeile \d{2}\.\d{3}
  2. Zeile: c:\windows`
```

Übrigens kann ein Raw String Literal auch von mehrfachen Backticks umschlossen sein (``*Enthält ` im Inhalt*``) und ermöglicht damit auch die Verwendung von Backticks im Text, ohne sie zu escapen.

Fazit

Die Idee der halbjährlichen Major Releases und damit regelmäßigen Updates scheint aufzugehen. Oracle hat bisher pünktlich geliefert, wir können wir auf die nächsten Neuerungen im Oktober 2019 gespannt sein. Nicht vergessen darf man, dass das JDK 12 seitens Oracle nur für ein halbes Jahr mit Updates unterstützt wird. Man hat also jetzt die Wahl und bleibt entweder bis 2021 beim JDK 8 oder dem letzten LTS-Release JDK 11. Alternativ kann man die halbjährlichen Releases mitgehen,

Listing 6

```
public static String describeInt(int i) {
  return switch (i) {
    case 1, 2 -> "one or two";
    case 3 -> "three";
    default -> "smaller than one or more than three";
  };
}
```

Listing 7

```
System.out.println("foobar".indent(5)); // '     foobar'

final Integer result = "42".<Integer>transform(s -> Integer.valueOf(s));

final long mismatch = Files.mismatch(Path.of("a.out"), Path.of("b.out"));
```

muss zwar häufig migrieren, vermeidet aber die aufwendigen Big-Bang-Umstiege auf eine neue Java-Version nach vielen Jahren. Zudem erhält man dadurch mit dem Oracle OpenJDK auch weiterhin kostenlose Updates. Denn sowohl das Oracle JDK 8 als auch 11 sind mittlerweile in Produktion nur noch kostenpflichtig einsetzbar. Allerdings findet man auch von anderen Anbietern wie AdoptOpenJDK, Amazon Corretto, Azul, Red Hat, IBM usw. sowohl freie als auch kommerzielle Alternativen mit Long Term Support, die über mehrere Jahre regelmäßig Sicherheitspatches zur Verfügung stellen wollen.

Egal, welche Updatestrategie man zukünftig anwenden möchte, es empfiehlt sich, das JDK 12 bereits jetzt herunterzuladen [4] und sich mit den Neuerungen, insbesondere den Switch Expressions, vertraut zu machen.

Falk Sippach hat zwanzig Jahre Erfahrung mit Java und ist bei der Mannheimer Firma OIO Orientation in Objects GmbH als Trainer, Softwareentwickler und -architekt tätig. Er publiziert regelmäßig in Blogs, Fachartikeln und auf Konferenzen. In seiner Wahlheimat Darmstadt organisiert er mit anderen die örtliche Java User Group.

@sippsack

Links & Literatur

[1] https://openjdk.java.net/projects/jdk/12/

[2] https://wiki.openjdk.java.net/display/zgc/Main

[3] https://github.com/AdoptOpenJDK/jdk-api-diff

[4] https://jdk.java.net/12/

Autor des Monats

Falk Sippach
OIO Orientation in Objects GmbH

Wie bist du zur Softwareentwicklung gekommen?

Eigentlich relativ spät. Viele aus meiner Generation haben schon als Kind mit C64, Amiga oder Atari gespielt und kleine BASIC-Programme abgetippt. Da hatten wir nicht mal einen langweiligen PC zu Hause. In der zehnten Klasse durfte ich dann während einer Projektwoche bei meinem späteren Informatiklehrer LEDs über die parallele Schnittstelle zum Leuchten bringen. Und ab da ging es los, zunächst mit Turbo Pascal im Informatikunterricht bzw. zur Ansteuerung der heimischen Modelleisenbahn und im Studium dann mit Java, PHP und diversen Webtechnologien. Die Faszination Softwareentwicklung hat mich bis heute nicht losgelassen.

Was ist für dich der schönste Aspekt in der Softwareentwicklung?

Ich bin immer wieder stolz, wenn ich eine elegante Lösung für ein Problem gefunden habe. Das mag noch so trivial sein, wenn es dem Benutzer Zeit und unnötige Arbeit erspart, macht mich das froh. Dank der großartigen Java-Community kann ich als wissbegieriger Mensch zudem täglich Neues lernen. Durch meine Veröffentlichungen und Vorträge versuche ich der Gemeinschaft etwas zurückzugeben.

Was ist für dich ein weniger schöner Aspekt?

Mir ist die ganze Technologiewelt zu schnelllebig geworden. Einerseits macht es die Arbeit in unserer Branche zwar sehr spannend, mich frustriert es aber letztlich, wenn ich wegen meiner begrenzten Zeit viele interessante Themen nicht näher anschauen kann. Trotzdem glaube ich weiter an den ständigen Lernprozess und gebe mir alle Mühe, mich nicht abhängen zu lassen.

Wie und wann bist du auf Java gestoßen?

Das war bereits in den Anfangsjahren, so etwa um das Jahr 1998. In meinem Studienjahrgang wurden die grundlegenden Programmierkurse gerade von C/C++ auf Java umgestellt. Besonders fasziniert hat mich damals die Programmierung grafischer Oberflächen mit AWT und Swing. Spannend finde ich das Java-Ökosystem bis heute, auch wenn sich zwischenzeitlich mein Fokus eher in Richtung Architektur, Backend und Webtechnologien verschoben hat.

Wenn du für einen Tag König der Java-Welt wärst, was würdest du verändern?

Natürlich gibt es einige Schwachstellen, aber tatsächlich bin ich mit der aktuellen Entwicklung von Java relativ zufrieden. Bei Java SE ist das Open-Source-Projekt OpenJDK nun die Basis für das JDK von Oracle und anderen Anbietern (Azul, Red Hat, Amazon …). Bei Java EE geht die Transition zur Eclipse Foundation zwar langsam, aber stetig voran. Ich glaube fest daran, dass wir in den nächsten Jahren noch viel Freude mit Java und den alternativen Sprachen auf der Java-Plattform haben werden.

Was ist zurzeit dein Lieblingsbuch?

Zwar stapeln sich auf meinem Nachtisch diverse IT-Bücher, (vor)lesen tue ich aktuell allerdings nur Kinderbücher wie „Ritter Tobi“ oder „Neues vom Mövenweg“. Sehr gespannt bin ich aber auf die Neuauflage von Martin Fowlers Refactoring-Buch. Zum Glück muss man da nur etwa hundert Seiten lesen, die restlichen Kapitel sind als Katalog zum Nachschlagen gedacht.

Was machst du in deinem anderen Leben?

Ich versuche möglichst viel Zeit mit meiner Familie zu verbringen, meine drei Kinder halten mich dabei ordentlich auf Trab. Als Abwechselung zur sitzenden Bürotätigkeit betätige ich mich zudem gern an der frischen Luft, sowohl zu Fuß als auch mit dem Rad oder im Winter auf Skiern. Dazu engagiere ich mich in diversen Sportvereinen als Übungsleiter.

) Seminarprogramm)

2. Halbjahr 2019

part of trivadis group

Java Seminare

Entscheider

- **Java für Entscheider** — Dauer: 1 Tag / Preis: € 825,- — Termine: 01.07. / 25.10. / 09.12.
- **Java im Web für Architekten** — Dauer: 2 Tage / Preis: € 1.500,- — Termine: 02.09. / 11.11.
- **NoSQL mit Java im Überblick** — Dauer: 2 Tage / Preis: € 1.500,- — Termine: 17.10. / 05.12.
- **Enterprise Java für Architekten** — Dauer: 2 Tage / Preis: € 1.500,- — Termine: 16.09. / 14.11.

Grundlagen

- **Java für Programmierer** — Dauer: 5 Tage / Preis: € 2.180,- — Termine: 01.07. / 07.10. / 09.12.
- **Einführung in Groovy** — Dauer: 3 Tage / Preis: € 1.730,- — Termin: 28.10.
- **Kotlin** — Dauer: 2 Tage / Preis: € 1.230,- — Termine: a. A.

Vertiefung

- **Refactoring Workshop** — Dauer: 2 Tage / Preis: € 1.230,- — Termin: 26.09.
- **Effective Java** — Dauer: 2 Tage / Preis: € 1.225,- — Termin: 24.10.
- **Testen von Java Programmen** — Dauer: 3 Tage / Preis: € 1.730,- — Termine: 23.09. / 18.11.
- **Clean Code und Software Craftsmanship** — Dauer: 4 Tage / Preis: € 2.230,- — Termin: 02.09.
- **Design Patterns mit Java** — Dauer: 2 Tage / Preis: € 1.150,- — Termin: 21.11.
- **Java 8 Update - Lambda und Streams** — Dauer: 2 Tage / Preis: € 1.290,- — Termine: 16.09 / 25.11.
- **Java 9 - 11 Update** — Dauer: 2 Tage / Preis: € 1.290,- — Termine: 12.09. / 05.12.
- **Reaktive Programmierung mit Java** — Dauer: 1 Tag / Preis: € 710,- — Termine: 23.10. / 10.12.
- **Resilient Software Design mit Java** — Dauer: 1 Tag / Preis: € 710,- — Termine: 25.10. / 09.12.

Java Enterprise

- **Jakarta EE Power Workshop** — Dauer: 5 Tage / Preis: € 2.560,- — Termine: 16.09. / 11.11.
- **Enterprise JavaBeans** — Dauer: 3 Tage / Preis: € 1.730,- — Termine: a. A.
- **Java Performance** — Dauer: 2 Tage / Preis: € 1.290,- — Termine: 08.07. / 21.10.
- **JavaFX für moderne GUI-Clients** — Dauer: 4 Tage / Preis: € 2.230,- — Termin: 28.10.
- **Java Persistence API (JPA) mit Hibernate** — Dauer: 3 Tage / Preis: € 1.730,- — Termine: 23.09. / 18.11.
- **Java Persistence Performance Tuning** — Dauer: 2 Tage / Preis: € 1.500,- — Termin: 23.10.
- **JavaServer Faces** — Dauer: 3 Tage / Preis: € 1.730,- — Termine: 14.10. / 02.12.
- **Microservices mit Java** — Dauer: 2 Tage / Preis: € 1.500,- — Termine: 07.10. / 25.11.
- **Java EE Batch Applications** — Dauer: 2 Tage / Preis: € 1.290,- — Termin: 02.09.
- **Web Services mit SOAP und Java** — Dauer: 3 Tage / Preis: € 1.730,- — Termin: 28.10.
- **OSGi Service Platform** — Dauer: 2 Tage / Preis: € 1.500,- — Termine: a. A.

Open Source Seminare

Software Entwicklungsumgebung

- **Versionsverwaltung mit Subversion** — Dauer: 1 Tag / Preis: € 710,- — Termine: a. A..
- **Git kompakt** — Dauer: 1 Tag / Preis: € 710,- — Termine: 10.10. / 28.11.
- **Versionsverwaltung mit Git** — Dauer: 2 Tage / Preis: € 1.290,- — Termin: 11.07.
- **Reporting mit Eclipse BIRT** — Dauer: 2 Tage / Preis: € 1.290,- — Termine: a. A..
- **Das Buildtool Apache Maven** — Dauer: 1 Tag / Preis: € 710,- — Termin: 30.09.
- **Gradle für Java Builds** — Dauer: 1 Tag / Preis: € 710,- — Termine: 11.10. / 29.11.
- **Jenkins Grundlagen** — Dauer: 1 Tag / Preis: € 825,- — Termin: 09.10.
- **Selenium - Test von Webanwendungen** — Dauer: 2 Tage / Preis: € 1.290,- — Termine: 26.09. / 21.11.

Server / Frameworks

- **Entwicklung und Betrieb mit Wildfly 10** — Dauer: 3 Tage / Preis: € 2.110,- — Termin: 04.09.
- **Einführung in das Spring Framework** — Dauer: 3 Tage / Preis: € 1.730,- — Termin: 02.12.
- **Spring Boot** — Dauer: 2 Tage / Preis: € 1.290,- — Termine: 02.07. / 21.10. / 10.12.
- **Cloud Native mit Spring** — Dauer: 2 Tage / Preis: € 1.290,- — Termine: 11.07. / 23.10. / 12.12.
- **Docker kompakt** — Dauer: 1 Tag / Preis: € 710,- — Termine: 09.10. / 27.11.
- **Kubernetes** — Dauer: 2 Tage / Preis: € 1.290,- — Termine: 10.10. / 28.11.
- **Einführung in GWT** — Dauer: 2 Tage / Preis: € 1.290,- — Termine: a. A.
- **GWT Architekturen** — Dauer: 2 Tage / Preis: € 1.500,- — Termine: a. A.
- **Apache Camel** — Dauer: 2 Tage / Preis: € 1.290,- — Termine: a. A.
- **Apache Cassandra** — Dauer: 2 Tage / Preis: € 1.290,- — Termine: a. A.
- **Apache Kafka** — Dauer: 2 Tage / Preis: € 1.290,- — Termine: a. A.

XML Seminare

Grundlagen

- **XML Einführung** — Dauer: 3 Tage / Preis: € 1.620,- — Termin: 08.07.
- **XML Schema** — Dauer: 2 Tage / Preis: € 1.230,- — Termine: a. A.

Vertiefung

- **XSL und Formatting Objects** — Dauer: 2 Tage / Preis: € 1.120,- — Termine: a. A.
- **Transformation und Styling mit XSLT** — Dauer: 2 Tage / Preis: € 1.230,- — Termin: 11.07.

Seminare zur Web-Programmierung

Programmierung

- **JavaScript Intensiveinstieg** — Dauer: 2 Tage / Preis: € 1.230,- — Termine: 04.07. / 12.09. / 07.11.
- **JavaScript Engineering** — Dauer: 2 Tage / Preis: € 1.230,- — Termine: 15.07. / 16.09. / 18.11.
- **Testen mit JavaScript** — Dauer: 1 Tag / Preis: € 675,- — Termine: 18.09. / 20.11.
- **ECMAScript mit Typescript** — Dauer: 3 Tage / Preis: € 1.730,- — Termin: 11.11.
- **Progressive Web Apps** — Dauer: 1 Tag / Preis: € 710,- — Termine: 26.09. / 14.11.
- **Angular** — Dauer: 3 Tage / Preis: € 1.730,- — Termine: 14.10. / 25.11.
- **Advanced Angular** — Dauer: 2 Tage / Preis: € 1.290,- — Termine: 17.10. / 28.11.
- **Webentwicklung mit React** — Dauer: 2 Tage / Preis: € 1.290,- — Termine: 17.10. / 05.12.

Design

- **Modernes Webdesign mit HTML 5 und CSS 3** — Dauer: 3 Tage / Preis: € 1.730,- — Termine: 01.07. / 09.09. / 04.11.
- **HTML 5 Update** — Dauer: 1 Tag / Preis: € 710,- — Termine: 01.07. / 09.09. / 04.11.
- **Modernes Webdesign mit CSS 3** — Dauer: 2 Tage / Preis: € 1.150,- — Termine: 02.07. / 10.09. / 05.11.
- **Web Components Grundlagen** — Dauer: 3 Tage / Preis: € 1.815,- — Termine: a. A.

Entscheider Seminare

Architekten

- **Java im Web für Architekten** — Dauer: 2 Tage / Preis: € 1.500,- — Termine: 02.09. / 11.11.
- **NoSQL mit Java im Überblick** — Dauer: 2 Tage / Preis: € 1.500,- — Termine: 17.10. / 05.12.
- **Architektur-Katas zum Training agiler Teams** — Dauer: 1 Tag / Preis: € 825,- — Termine: 18.09. / 13.11.
- **Enterprise Java für Architekten** — Dauer: 2 Tage / Preis: € 1.500,- — Termine: 16.09. / 14.11.
- **Microservices für Entscheider** — Dauer: 1 Tag / Preis: € 825,- — Termine: a. A.
- **API Design für Architekten** — Dauer: 1 Tag / Preis: € 825,- — Termine: 27.09. / 15.11..
- **Architektur-Dokumentation** — Dauer: 2 Tage / Preis: € 1.500,- — Termin: 30.09.

Projektleiter

- **Java für Entscheider** — Dauer: 1 Tag / Preis: € 825,- — Termine: 01.07. / 25.10. / 09.12.
- **Führen und Managen von Projektteams** — Dauer: 2 Tage / Preis: € 1.500,- — Termin: 16.12.
- **Grundlagen Projektmanagement** — Dauer: 2 Tage / Preis: € 1.500,- — Termin: 04.09.
- **Open Source Compliance Management kompakt** — Dauer: 1 Tag / Preis: € 825,- — Termine: 13.09. / 08.11.

Seminare zur Methodik und Prozessen

Methodik

- **SAFe® for Teams** — Dauer: 2 Tage / Preis: € 1.595,- — Termine: a. A.
- **Leading SAFe®** — Dauer: 2 Tage / Preis: € 1.595,- — Termine: 19.09. / 21.11.
- **Scrum Jumpstart** — Dauer: 1 Tag / Preis: € 825,- — Termine: 11.09. / 06.11.
- **Kanban Jumpstart** — Dauer: 1 Tag / Preis: € 825,- — Termine: 12.09. / 07.11.
- **User Stories effektiv erstellen** — Dauer: 1 Tag / Preis: € 825,- — Termine: 09.09. / 04.11.
- **Domain Driven Design** — Dauer: 1 Tag / Preis: € 710,- — Termin: 02.10.
- **Kaikaku und Kaizen in der Softwareentwicklung** — Dauer: 1 Tag / Preis: € 710,- — Termine: a. A.
- **UML für Analytiker** — Dauer: 3 Tage / Preis: € 1.730,- — Termine: a. A.
- **Architektur-Katas zum Training agiler Teams** — Dauer: 1 Tag / Preis: € 825,- — Termine: 18.09. / 13.11.
- **Grundlagen Projektmanagement** — Dauer: 2 Tage / Preis: € 1.500,- — Termin: 04.09.

Soft Skills

- **Soft Skills für Agile Projekte** — Dauer: 2 Tage / Preis: € 1.500,- — Termin: 02.09.
- **Persönlichkeitsentwicklung** — Dauer: 2 Tage / Preis: € 1.500,- — Termin: 30.09.
- **Zeit- und Selbstmanagement** — Dauer: 1 Tag / Preis: € 825,- — Termin: 02.10.
- **Führen und Managen von Projektteams** — Dauer: 2 Tage / Preis: € 1.500,- — Termin: 16.12.

Werkzeuge

- **Confluence für Anwender** — Dauer: 1 Tag / Preis: € 825,- — Termine: 10.09. / 05.11.
- **Confluence - Fachliche Administration** — Dauer: 1 Tag / Preis: € 825,- — Termine: 11.09. / 06.11.
- **Confluence - Macros** — Dauer: 1 Tag / Preis: € 825,- — Termin: 02.10.
- **Bitbucket Server - Enterprise Git** — Dauer: 2 Tage / Preis: € 1.500,- — Termine: 21.10. / 12.12.
- **Jira Plattform - Fachliche Administration** — Dauer: 1 Tag / Preis: € 825,- — Termine: 12.09. / 07.11.
- **Jira Software für agile Projekte** — Dauer: 1 Tag / Preis: € 825,- — Termine: 13.09. / 08.11.
- **Jira for Business** — Dauer: 1 Tag / Preis: € 825,- — Termine: 09.09. / 04.11.
- **Jira Service Desk** — Dauer: 1 Tag / Preis: € 825,- — Termine: 10.09. / 05.11.

Alle Seminare sind auch als Inhouse- oder Individualschulungen möglich.

* Alle Preise verstehen sich zzgl. gesetzl. Mehrwertsteuer.
Änderungen vorbehalten.

© Orientation in Objects GmbH | Weinheimer Str. 68 | 68309 Mannheim | Tel. +49 621 71839-0 | Fax +49 621 71839-50 | info@oio.de | www.oio.de

OIO Orientation in Objects GmbH

Weinheimer Str. 68
68309 Mannheim
Telefon: +49 621 71839-0
Fax: +49 621 71839-50
Mail: info@oio.de