Michael Inden

Java

Die Neuerungen in Version 9 bis 14

Modularisierung, Syntax- und API-Erweiterungen

dpunkt.verlag

Dipl.-Inform. Michael Inden ist Oracle-zertifizierter Java-Entwickler. Nach seinem Studium in Oldenburg hat er bei diversen internationalen Firmen in verschiedenen Rollen etwa als Softwareentwickler, -architekt, Consultant, Teamleiter sowie Trainer gearbeitet. Zurzeit ist er als CTO und Leiter Academy in Zürich tätig.

Michael Inden hat über zwanzig Jahre Berufserfahrung beim Entwurf komplexer Softwaresysteme gesammelt, an diversen Fortbildungen und mehreren Java-One-Konferenzen teilgenommen. Sein besonderes Interesse gilt dem Design qualitativ hochwertiger Applikationen mit ergonomischen GUIs sowie dem Coaching. Sein Wissen gibt er gerne als Trainer in internen und externen Schulungen und auf Konferenzen weiter, etwa bei der Java User Group Switzerland, bei der JAX/W-JAX, ch.open, der Oracle Code One und den IT-Tagen.

Michael Inden

Java – die Neuerungen in Version 9 bis 14

Modularisierung, Syntax- und API-Erweiterungen

dpunkt.verlag

Michael Inden
michael_inden@hotmail.com

Lektorat: Dr. Michael Barabas
Copy-Editing: Ursula Zimpfer, Herrenberg
Satz: Michael Inden
Herstellung: Stefanie Weidner
Umschlaggestaltung: Helmut Kraus, *www.exclam.de*
Druck und Bindung: mediaprint solutions GmbH, 33100 Paderborn

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über *http://dnb.d-nb.de* abrufbar.

ISBN:

Print	978-3-86490-754-8
PDF	978-3-96088-978-6
ePub	978-3-96088-979-3
mobi	978-3-96088-980-9

1. Auflage 2020

Wieblinger Weg 17
69123 Heidelberg

Hinweis:
Dieses Buch wurde auf PEFC-zertifiziertem Papier aus nachhaltiger Waldwirtschaft gedruckt. Der Umwelt zuliebe verzichten wir zusätzlich auf die Einschweißfolie.

Schreiben Sie uns:
Falls Sie Anregungen, Wünsche und Kommentare haben, lassen Sie es uns wissen: hallo@dpunkt.de.

5 4 3 2 1 0

Inhaltsverzeichnis

III Modularisierung

Vorwort

Zunächst einmal bedanke ich mich bei Ihnen, dass Sie sich für dieses Buch entschieden haben. Hierin finden Sie eine Vielzahl an Informationen zu den Neuerungen in der aktuellen Java-Version 14 und in den Vorgängern 9 bis 13. Aufgrund des mittlerweile halbjährlichen Releasezyklus sind in den Java-Versionen 10, 11, 12 und 13 jeweils weniger Änderungen als in früheren Releases enthalten. In diesem Buch werden insbesondere auch diverse Neuerungen aus Java 9 beschrieben, weil Java 9 das letzte große Update nach Java 8 war und eine Vielzahl an relevanten Erweiterungen mitbringt.

Zielgruppe

Dieses Buch ist kein Buch für Programmierneulinge, sondern richtet sich an diejenigen Leser, die bereits solides Java-Know-how besitzen und sich kurz und prägnant über die wichtigsten Neuerungen in den Java-Versionen 9 bis 14 informieren wollen. Dieses Buch wendet sich im Speziellen an zwei Zielgruppen:

1. Zum einen sind dies engagierte Hobbyprogrammierer und Informatikstudenten, aber auch Berufseinsteiger, die Java als Sprache beherrschen und an den Neuerungen in Java 9 bis 14 interessiert sind.
2. Zum anderen ist das Buch für erfahrene Softwareentwickler und -architekten gedacht, die ihr

Wissen ergänzen oder auffrischen wollen, um für zukünftige Projekte abschätzen zu können, ob und – wenn ja – für welche Anforderungen die neuen Java-Versionen eine gewinnbringende Alternative darstellen können.

Was vermittelt dieses Buch?

Sie als Leser erhalten in diesem Buch neben Theoriewissen eine Vertiefung durch praktische Beispiele, sodass der Umstieg auf Java 9 bis 14 in eigenen Projekten erfolgreich gemeistert werden kann. Erleichtert wird ein Umstieg durch eine Vielzahl an Übungen, die dem Leser die jeweiligen Features dediziert nahebringen.

Um die Beispiele des Buchs möglichst präzise und elegant zu halten, verwende ich diverse Features aus Java 8. Deshalb ist es hilfreich, wenn Sie sich damit schon beschäftigt haben. Alle, die eine kleine Auffrischung benötigen, finden zum leichteren Einstieg in Anhang A einen Crashkurs zu Java 8.

Aufbau dieses Buchs

Dieses Buch besteht in Wesentlichen aus zwei größeren Teilbereichen, nämlich einem zu den Neuerungen in Java 9 bis 11 und einem zu den Neuerungen in Java 12 bis 14. Diese Untergliederung habe ich gewählt, weil Java 9 und 10 nicht mehr ohne spezielle Tricks als Download erhältlich sind und Java 11 ein über Jahre unterstütztes Release, ein sogenanntes LTS-Release, darstellt, wobei LTS für Long Term Support steht. Die aktuellsten Entwicklungen sind dann für Java 12 bis 14 beschrieben. Nachfolgend möchte ich die Themen der einzelnen Kapitel kurz vorstellen.

Kapitel 1 – Einleitung Die Einleitung stimmt Sie auf Java 9 bis 14 ein und gibt einen Überblick, was Sie so alles in diesem Buch bzw. als Neuerungen erwartet.

Kapitel 2 – Syntaxerweiterungen in JDK 9 bis 11 Zunächst widmen wir uns verschiedenen Änderungen an der Syntax von Java. Neben Details zu Bezeichnern, dem Diamond Operator und Ergänzungen bei Annotations gehe ich vor allem kritisch auf das Feature privater Methoden in Interfaces ein. Für Java 10 und 11 stelle ich `var` als Möglichkeit zur Definition lokaler Variablen bzw. zur Verwendung in Lambdas vor.

Kapitel 3 – Neues und Änderungen in JDK 9 In den APIs des JDKs finden sich diverse Neuerungen. Dieses Potpourri habe ich thematisch ein wenig gegliedert. Neben Vereinfachungen beim Prozess-Handling, der Verarbeitung mit `Optional<T>` oder von Daten mit `InputStreams` schauen wir auf fundamentale Neuerungen im Bereich der Concurrency durch Reactive Streams.

Kapitel 4 – Neues und Änderungen in Java 10 Seit Java 10 verfolgt man bei Oracle die Strategie, Java in kleinen, aber feinen Iterationen um nützliche Funktionalität zu ergänzen und auch in Bezug auf die Syntax zu modernisieren. In diesem Kapitel werden einige API-Erweiterungen vorgestellt.

Kapitel 5 – Neues und Änderungen in Java 11 Neben einigen API-Erweiterungen wurden vor allem kleinere Bereinigungen im JDK vorgenommen, unter anderem sind die Module zu CORBA, JavaFX und in Teilen auch zu XML, speziell JAXB, nun nicht mehr Bestandteil des JDKs.

Kapitel 6 – JVM-Änderungen in JDK 9 bis 11 In diesem Kapitel beschäftigen wir uns mit Änderungen in der JVM, etwa bei der Garbage Collection oder der Einführung der `jshell`. Auch in Bezug auf `javadoc` und der Nummerierung von Java-Versionen finden wir in Java 9 Änderungen, die thematisiert werden. Java 11 bringt einen neuen Garbage Collector sowie mit dem Feature »Launch Single-File Source-Code Programs« die Möglichkeit, Java-Klassen ohne explizite vorherige Kompilierung ausführen zu können.

Kapitel 7 – Übungen zu den Neuerungen in JDK 9 bis 11 Zur Vertiefung des Gelernten präsentiert dieses Kapitel diverse Übungsaufgaben zu den Themen der vorangegangenen Kapitel 2 bis 6.

Kapitel 8 – Neues und Änderungen in Java 12 Für uns als Entwickler relevante Neuerungen finden sich in Java 12 kaum. Erwähnenswert ist vor allem ein Preview auf Syntaxänderungen bezüglich `switch`. Darüber hinaus könnte Sie das Microbenchmark-Framework JMH (Java Microbenchmarking Harness) interessieren, sofern Sie Optimierungen auf Mikroebene vornehmen mussten, um das letzte Quäntchen an Performance herauszuholen. Ergänzend bietet Java 12 dann noch kleinere API-Neuerungen.

Kapitel 9 – Neues und Änderungen in Java 13 und 14 Java 13 umfasst vor allem zwei Previews auf Syntaxänderungen: Zum einen ein Folge-Preview zu `switch` und zum anderen ein Preview auf mehrzeilige Strings, »Text Blocks« genannt. Da diese auch in Java 14 enthalten sind, existiert kein gesondertes Kapitel zu Java 13, sondern alles wird in diesem Kapitel beschrieben. Mit Java 14 werden die Neuerungen bei `switch` endlich in den Sprachstandard aufgenommen. Zudem gibt es eine

hilfreiche Ergänzung zur Fehleranalyse bei `NullPointerExceptions`. Darüber hinaus finden wir unter anderem folgende Preview-Features: Sogenannte Records, die eine extrem kompakte Schreibweise zum Deklarieren spezieller Klassen mit unveränderlichen Daten bieten und die automatisch das passende API bereitstellen. Im Kontext von `instanceof` erlaubt es eine Syntaxerweiterung, unschöne Casts und künstliche Hilfsvariablen zu vermeiden. Zudem bietet Java 14 zwei Erweiterungen bei der Definition mehrzeiliger Texte.

Kapitel 10 – Übungen zu den Neuerungen in JDK 12 bis 14 Auch für die Neuerungen aus Java 12 bis 14 wird ein umfangreicher Satz an Übungsaufgaben zu den Themen der vorangegangenen Kapitel 8 und 9 präsentiert. Deren Bearbeitung sollte Ihr Wissen dazu vertiefen.

Kapitel 11 – Modularisierung mit Project Jigsaw Klar strukturierte Softwarearchitekturen mit sauber definierten Abhängigkeiten sind erstrebenswert, um selbst größere Softwaresysteme möglichst beherrschbar zu machen und Teile unabhängig voneinander änderbar zu halten. Seit Java 9 helfen dabei Module als eigenständige Softwarekomponenten. In diesem Kapitel wird die Thematik Modularisierung eingeführt und anhand von Beispielen vorgestellt. Im Speziellen werden Themen wie Sichtbarkeit und Zugriffsschutz behandelt.

Kapitel 12 – Weiterführende Themen zur Modularisierung In diesem Kapitel gehe ich auf einige fortgeschrittenere Themen zur Modularisierung ein. Zunächst stelle ich Ihnen eine Alternative zum von Oracle propagierten, aber in der Praxis hinderlichen Verzeichnislayout für Module vor. Danach betrachten wir die Abhängigkeitssteuerung in größerer Tiefe: Zwar hilft

die Modularisierung bei der Strukturierung eines Systems, jedoch besitzen die Module oftmals direkte Abhängigkeiten bereits zur Kompilierzeit. Wird eine losere Kopplung benötigt, so kann man dafür Services nutzen. Zudem ändern sich durch die Modularisierung ein paar Dinge bezüglich Reflection, beispielsweise lassen sich neue Eigenschaften ermitteln, etwa die Moduldaten zu einer Klasse. Verbleibt noch ein wichtiges Thema, nämlich die Migration einer bestehenden Applikation in eine modularisierte. Weil dabei doch ein paar Dinge zu beachten sind, ist diesem Thema ein ausführlicher Abschnitt gewidmet, der insbesondere die verschiedenen Arten von Modulen und ihre Eigenschaften behandelt.

Kapitel 13 – Übungen zur Modularisierung Wie für die API-Erweiterungen werden auch für die Modularisierung verschiedene Übungsaufgaben in einem Kapitel zusammengestellt.

Kapitel 14 – Zusammenfassung Dieses Kapitel fasst die Themen rund um die vielfältigen Neuerungen aus Java 9 bis 14 noch einmal kurz zusammen.

Anhang A – Schnelleinsteg in Java 8 In Anhang A werden für dieses Buch wesentliche Ergänzungen aus Java 8 rekapituliert. Das erleichtert Ihnen das Verständnis der Neuerungen in aktuellen Java-Versionen, selbst dann, wenn Sie sich noch nicht eingehend mit Java 8 beschäftigt haben. Neben einer Vorstellung der funktionalen Programmierung mit Lambdas widmen wir uns den Streams, einer wesentlichen Neuerung in JDK 8 zur Verarbeitung von Daten. Abgerundet wird Anhang A durch einen kurzen Blick auf das Date and Time API und verschiedene API-Erweiterungen.

Anhang B – Einführung Gradle Anhang B liefert eine kurze Einführung in das Build-Tool Gradle, das auch für die Beispiele dieses Buchs zur Übersetzung genutzt wird. Mithilfe des vermittelten Wissens sollten Sie dann auch kleinere eigene Projekte mit einem Build-System ausstatten können.

Anhang C – Einführung Maven In diesem Anhang wird Maven als Build-Tool kurz vorgestellt. Derzeit bietet es die beste Unterstützung für modularisierte Applikationen in Java. Zudem kann man Maven-Projekte einfach in gängige IDEs importieren.

Sourcecode und ausführbare Programme

Um den Rahmen des Buchs nicht zu sprengen, stellen die Listings häufig nur Ausschnitte aus lauffähigen Programmen dar, wobei wichtige Passagen zum besseren Verständnis mitunter fett hervorgehoben sind. Auf der Webseite zu diesem Buch `www.dpunkt.de/java-9-14-die-neuerungen` steht dann der vollständige, kompilierbare Sourcecode zu den Programmen zum Download bereit. Neben dem Sourcecode befindet sich auf der Webseite auch ein Eclipse-Projekt, über das sich alle Programme ausführen lassen.

Ergänzend wird die Datei `build.gradle` mitgeliefert, die den Ablauf des Builds für Gradle beschreibt. Dieses Build-Tool besitzt viele Vorzüge, wie die kompakte und gut lesbare Notation, und vereinfacht die Verwaltung von Abhängigkeiten enorm. Darüber hinaus erlaubt Gradle das Starten von Programmen, wobei der jeweilige

Programmname in Kapitälchenschrift, etwa DATETIMEEXAMPLE, angegeben wird.

Blockkommentare in Listings Beachten Sie bitte, dass sich in den Listings diverse Blockkommentare finden, die der Orientierung und dem besseren Verständnis dienen. In der Praxis sollte man derartige Kommentierungen mit Bedacht einsetzen und lieber einzelne Sourcecode-Abschnitte in Methoden auslagern. Für die Beispiele des Buchs dienen diese Kommentare aber als Anhaltspunkte, weil die eingeführten oder dargestellten Sachverhalte für Sie als Leser vermutlich noch neu und ungewohnt sind.

```
public static void main(final String[] args) throws
InterruptedException,

                        IOException

{

    // Prozess erzeugen

    final String command = "sleep 60s";

    final String commandWin = "cmd timeout 60";

    final Process sleeper =
    Runtime.getRuntime().exec(command);

    ...


    // Process => ProcessHandle

    final ProcessHandle sleeperHandle =
    ProcessHandle.of(sleeper.pid()).

                        orElseThrow(IllegalStateException::n
                        ew);
```

```
    ...

}
```

Konventionen

Verwendete Zeichensätze

In diesem Buch gelten folgende Konventionen bezüglich der Schriftart: Neben der vorliegenden Schriftart werden wichtige Textpassagen *kursiv* oder ***kursiv und fett*** markiert. Englische Fachbegriffe werden eingedeutscht großgeschrieben, etwa Event Handling. Zusammensetzungen aus englischen und deutschen (oder eingedeutschten) Begriffen werden mit Bindestrich verbunden, z. B. Plugin-Manager. Namen von Programmen und Entwurfsmustern werden in KAPITÄLCHEN geschrieben. Listings mit Sourcecode sind in der Schrift `Courier` gesetzt, um zu verdeutlichen, dass dies einen Ausschnitt aus einem Java-Programm darstellt. Auch im normalen Text wird für Klassen, Methoden, Konstanten und Parameter diese Schriftart genutzt.

Tipps und Hinweise aus der Praxis

Dieses Buch ist mit diversen Praxistipps gespickt. In diesen werden interessante Hintergrundinformationen präsentiert oder es wird auf Fallstricke hingewiesen.

Tipp: Praxistipp

In derart formatierten Kästen finden sich im späteren Verlauf des Buchs immer wieder einige wissenswerte Tipps und ergänzende Hinweise zum eigentlichen Text.

Verwendete Klassen aus dem JDK

Werden Klassen des JDKs erstmalig im Text erwähnt, so wird deren voll qualifizierter Name, d. h. inklusive der Package-Struktur, angegeben: Die Klasse `String` würde demnach als `java.lang.String` notiert – alle weiteren Nennungen erfolgen dann ohne Angabe des Package-Namens. Diese Regelung erleichtert initial die Orientierung und ein Auffinden im JDK und zudem wird der nachfolgende Text nicht zu sehr aufgebläht. Die voll qualifizierte Angabe hilft insbesondere, da in den Listings eher selten `import`-Anweisungen abgebildet werden.

Im Text beschriebene Methodenaufrufe enthalten in der Regel die Typen der Übergabeparameter, etwa `substring(int, int)`. Sind die Parameter in einem Kontext nicht entscheidend, wird mitunter auf deren Angabe aus Gründen der besseren Lesbarkeit verzichtet – das gilt ganz besonders für Methoden mit generischen Parametern.

Verwendete Abkürzungen

Im Buch verwende ich die in der nachfolgenden Tabelle aufgelisteten Abkürzungen. Weitere Abkürzungen werden im laufenden Text in Klammern nach ihrer ersten Definition aufgeführt und anschließend bei Bedarf genutzt.

Abkürzung	Bedeutung
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
(G)UI	(Graphical) User Interface
IDE	Integrated Development Environment
JDK	Java Development Kit
JEP	JDK Enhancement Proposal
JLS	Java Language Specification

JRE	Java Runtime Environment
JSR	Java Specification Request
JVM	Java Virtual Machine

Danksagung

Ein Fachbuch zu schreiben ist eine schöne, aber arbeitsreiche und langwierige Aufgabe. Alleine kann man dies kaum bewältigen. Daher möchte ich mich an dieser Stelle bei allen bedanken, die direkt oder indirekt zum Gelingen des Buchs beigetragen haben. Insbesondere konnte ich bei der Erstellung des Manuskripts auf ein starkes Team an Korrekturlesern zurückgreifen. Es ist hilfreich, von den unterschiedlichen Sichtweisen und Erfahrungen profitieren zu dürfen.

Zunächst einmal möchte ich mich bei Michael Kulla, der als Trainer für Java SE und Java EE bekannt ist, für sein mehrmaliges, gründliches Review vieler Kapitel und die fundierten Anmerkungen bedanken. Ebenfalls bin ich Prof. Dr. Dominik Gruntz sehr dankbar für diverse Verbesserungsvorschläge in den Kapiteln zu Java 12 und 14. Zudem erhielt ich den einen oder anderen Tipp von Jean-Claude Brantschen sowie noch in letzter Minute von Prof. Dr. Carsten Kern.

Nachfolgende Danksagung bezieht sich auf den Java-9-Teil sowie die Texte zur Modularisierung. Merten Driemeyer, Dr. Clemens Gugenberger, Prof. Dr. Carsten Kern sowie Andreas Schöneck haben mit verschiedenen hilfreichen Anmerkungen zu einer Verbesserung beigetragen. Zudem hat Ralph Willenborg dort mal wieder ganz genau gelesen und so diverse Tippfehler gefunden. Vielen Dank dafür! Auch von Albrecht Ermgassen erhielt ich den einen oder anderen Hinweis.

Schließlich bedanke ich mich bei einigen ehemaligen Arbeitskollegen der Zühlke Engineering AG: Jeton Memeti und Marius Reusch trugen durch ihre Kommentare zur Klarheit und Präzisierung bei. Auch von Hermann Schnyder von der Swisscom erhielt ich ein paar Anregungen.

Ebenso geht ein Dankeschön an das Team des dpunkt.verlags (Dr. Michael Barabas, Martin Wohlrab, Anja Weimer und Stefanie Weidner) für die tolle Zusammenarbeit. Außerdem möchte ich mich bei Torsten Horn für die fundierte fachliche Durchsicht sowie bei Ursula Zimpfer für ihre Adleraugen beim Copy-Editing bedanken.

Abschließend geht ein lieber Dank an meine Frau Lilija für ihr Verständnis und die Unterstützung. Glücklicherweise musste sie beim Entstehen dieses Buchs zu den Neuerungen in Java 9 bis 14 einen weit weniger gestressten Autor ertragen, als dies früher beim Schreiben meines Buchs »Der Weg zum Java-Profi« der Fall war.

Anregungen und Kritik

Trotz großer Sorgfalt und mehrfachen Korrekturlesens lassen sich missverständliche Formulierungen oder sogar Fehler leider nicht vollständig ausschließen. Falls Ihnen etwas Derartiges auffallen sollte, so zögern Sie bitte nicht, mir dies mitzuteilen. Gerne nehme ich auch Anregungen oder Verbesserungsvorschläge entgegen. Kontaktieren Sie mich bitte per Mail unter:

michael_inden@hotmail.com

Zürich, im März 2020
Michael Inden

1 Einleitung

Früher wurden Java-Releases aufgrund unfertiger Features häufiger verschoben. Um dem entgegenzuwirken, hat Oracle nach dem Erscheinen von Java 9 auf einen halbjährlichen Releasezyklus umgestellt. Das erlaubt es, die jeweils bis zu diesem Zeitpunkt fertig implementierten Funktionalitäten zu veröffentlichen. Zwar kann diese schnelle Releasefolge eine größere Herausforderung für Toolhersteller sein, für uns als Entwickler ist es aber oftmals positiv, weil wir potenziell weniger lang auf neue Features warten müssen. Das konnte früher recht mühsam sein, wie die letzten Jahre gezeigt haben. Ein paar Gedanken dazu greift der nachfolgende Hinweiskasten auf.

Allerdings gilt das Positive vor allem für eigene Hobbyprojekte, weil man dort mit den Neuerungen experimentieren kann und weniger durch Restriktionen eingeschränkt ist. Im professionellen Einsatz wird man eher auf Kontinuität und die Verfügbarkeit von Security Updates setzen, weshalb in diesem Kontext vermutlich nur LTS-Versionen in Betracht kommen, um Migrationsaufwände kalkulierbar und zeitlich besser planbar zu halten.

Hinweis: Oracles neue Releasepolitik

Bis einschließlich Java 9 wurden neue Java-Versionen immer Feature-basiert veröffentlicht. Das hatte in der Vergangenheit oftmals und mitunter auch beträchtliche Verschiebungen des geplanten Releasetermins zur Folge, wenn

für die Version wesentliche Features noch nicht fertig waren. Insbesondere deshalb verzögerten sich Java 8 und Java 9 um mehrere Monate bzw. sogar über ein Jahr: Rund 3,5 Jahre nach dem Erscheinen von JDK 8 im März 2014 ging Java mit Version 9 im September 2017 an den Start. Wieder einmal musste die Java-Gemeinde auf die Veröffentlichung der Version 9 des JDKs länger warten – es gab gleich mehrere Verschiebungen, zunächst von September 2016 auf März 2017, dann auf Juli 2017 und schließlich auf September 2017.

Mit einer zeitbasierten Releasestrategie möchte man derartigen Verzögerungen entgegenwirken, indem jedes halbe Jahr eine neue Java-Version veröffentlicht wird, die all jene Features enthält, die bereits fertig sind. Alle drei Jahre ist dann eine LTS-Version (Long Term Support) geplant. Eine solche ist in etwa vergleichbar mit den früheren Major-Versionen.

Was erwartet Sie im Folgenden?

Dieses Buch gibt einen fundierten Überblick über diverse wesentliche Erweiterungen in den JDKs 9 bis 14. Es werden unter anderem folgende Themen behandelt:

API- und Syntaxerweiterungen Wir schauen uns verschiedene Änderungen an der Syntax von Java an. Neben Erweiterungen bei der `@Deprecated`-Annotation widmen wir uns Details zu Bezeichnern, dem Diamond Operator und vor allem gehe ich kritisch auf das Feature privater Methoden in Interfaces ein. Für Java 10 und 11 thematisiere ich die Syntaxerweiterung `var` als Möglichkeit zur Definition lokaler Variablen bzw. zur Verwendung in Lambdas. Im Kontext von `instanceof` ist es mit Java 14 möglich, künstliche Hilfsvariablen und unschöne Casts zu vermeiden.

Kommen wir zu den APIs: In Java 9 wurden diverse APIs ergänzt oder neu eingeführt. Auch Bestehendes, wie z. B. das Stream-API oder die Klasse `Optional<T>`, wurde um Funktionalität erweitert. Neben Vereinfachungen beim Prozess-Handling, der Verarbeitung mit `Optional<T>` oder von Daten mit `InputStreams` schauen wir auf fundamentale

Neuerungen im Bereich der Concurrency durch Reactive Streams. Darüber hinaus enthalten Java 10 und 11 eine Vielzahl kleinerer weiterer Neuerungen. Eine größere Änderung ist der mit Java 11 offiziell ins JDK 11 aufgenommene HTTP/2-Support. In Java 12 wurden ein paar API-Erweiterungen integriert. Mit Java 13 finden wir zwei Previews auf Syntaxänderungen, nämlich einmal bezüglich `switch` und zudem die sogenannten »Text Blocks«, die mehrzeilige Strings erlauben. Mit Java 14 werden die Neuerungen bei `switch` endlich in den Sprachstandard aufgenommen. Außerdem finden wir eine hilfreiche Neuerung zur Fehleranalyse bei `NullPointerExceptions`. Zusätzlich bietet Java 14 zwei Erweiterungen bei der Definition mehrzeiliger Texte. Ganz besonders interessant sind sogenannte Records, die eine extrem kompakte Schreibweise zum Deklarieren spezieller Klassen mit unveränderlichen Daten ermöglichen.

JVM-Änderungen In jeweils eigenen Abschnitten beschäftigen wir uns mit Änderungen in der JVM, für JDK 9 etwa in Bezug auf die Nummerierung von Java-Versionen oder `javadoc`. Zudem kann für Quereinsteiger und Neulinge die durch das Tool `jshell` bereitgestellte Java-Konsole mit REPL-Unterstützung (Read-Eval-Print-Loop) erste Experimente und Gehversuche erleichtern, ohne dafür den Compiler oder eine IDE bemühen zu müssen. Mit Java 11 kommt ein neuer Garbage Collector sowie mit dem Feature »Launch Single-File Source-Code Programs« die Möglichkeit, Java-Klassen ohne explizite vorherige Kompilierung ausführen zu lassen und somit für Scripting einsetzen zu können. Java 12 bietet als wesentliche Neuerung die Integration des Microbenchmark-Frameworks JMH (Java Microbenchmarking Harness).

Modularisierung Die Modularisierung adressiert zwei typische Probleme größerer Java-Applikationen. Zum einen ist dies die sogenannte JAR-Hell, womit gemeint ist, dass sich im `CLASSPATH` verschiedene JARs mit zum Teil inhaltlichen Überschneidungen (unterschiedliche Versionen mit Abweichungen in Packages oder gleiche Klassen, jedoch mit anderem Bytecode) befinden. Dabei kann allerdings nicht sichergestellt werden, wann welche Klasse aus welchem JAR eingebunden wird. Zum anderen sind als `public` definierte Typen beliebig von anderen Packages aus zugreifbar. Das erschwert die Kapselung. Mit JDK 9 lassen sich eigenständige Softwarekomponenten (Module) mit einer Sichtbarkeitssteuerung definieren. Das hat allerdings weitreichende Konsequenzen: Sofern man Module verwendet, kann man Programme mit JDK 9 nicht mehr ohne Weiteres wie gewohnt starten, wenn diese externe Abhängigkeiten besitzen. Das liegt vor allem daran, dass Abhängigkeiten nun beim Programmstart geprüft und dazu explizit beschrieben werden müssen.

Es gibt aber einen rein auf dem `CLASSPATH` basierenden Kompatibilitätsmodus, der ein Arbeiten wie bis einschließlich JDK 8 gewohnt ermöglicht.

Tipp: Beispiele und der Kompatibilitätsmodus

Zum Ausprobieren verschiedener Neuerungen aus JDK 9 bis 14 werden wir kleine Beispielapplikationen in `main()`-Methoden erstellen. Dabei ist es für erste Experimente und für die Migration bestehender Anwendungen von großem Vorteil, dass man das an sich modularisierte JDK auch ohne eigene Module und ihre Sichtbarkeitsbeschränkungen betreiben kann. In diesem Kompatibilitätsmodus wird wie zuvor bei Java 8 mit `.class`-Dateien, JARs und dem `CLASSPATH` gearbeitet. Für zukünftige Projekte wird man mitunter Module nutzen wollen. Das schauen wir uns in eigenen Kapiteln an.

Ausprobieren der Java-14-Beispiele

Durch die kurzen Releasezyklen werden mittlerweile einige Features als Previews der Entwicklergemeinde vorgestellt. Möchte man diese nutzen, so sind in den IDEs und Build-Tools gewisse Parametrierungen sowohl beim Kompilieren als auch beim Ausführen nötig, etwa wie im folgenden Beispiel:

```
java --enable-preview -cp build/libs/Java14Examples.jar \

     java14.RecordExamples
```

Details dazu werden in Abschnitt 9.4 beschrieben.

Entdeckungsreise JDK 9 bis 14 – Wünsche an die Leser

Ich wünsche allen Lesern viel Freude mit diesem Buch sowie einige neue Erkenntnisse und viel Spaß beim Experimentieren mit JDK 9 bis 14. Möge Ihnen der Umstieg auf die von Ihnen bevorzugte Java-Version und die Erstellung modularer Applikationen oder die Migration bestehender Anwendungen durch die Lektüre meines Buchs leichter fallen.

Wenn Sie zunächst eine Auffrischung Ihres Wissens zu Java 8 und seinen Neuerungen benötigen, bietet sich ein Blick in den Anhang A an.