



# Java Übungsbuch

**Aufgaben mit vollständigen Lösungen**

**Für die Versionen Java 8 bis Java 17**



## **Hinweis des Verlages zum Urheberrecht und Digitalen Rechtemanagement (DRM)**

Liebe Leserinnen und Leser,

dieses E-Book, einschließlich aller seiner Teile, ist urheberrechtlich geschützt. Mit dem Kauf räumen wir Ihnen das Recht ein, die Inhalte im Rahmen des geltenden Urheberrechts zu nutzen. Jede Verwertung außerhalb dieser Grenzen ist ohne unsere Zustimmung unzulässig und strafbar. Das gilt besonders für Vervielfältigungen, Übersetzungen sowie Einspeicherung und Verarbeitung in elektronischen Systemen.

Je nachdem wo Sie Ihr E-Book gekauft haben, kann dieser Shop das E-Book vor Missbrauch durch ein digitales Rechtemanagement schützen. Häufig erfolgt dies in Form eines nicht sichtbaren digitalen Wasserzeichens, das dann individuell pro Nutzer signiert ist. Angaben zu diesem DRM finden Sie auf den Seiten der jeweiligen Anbieter.

Beim Kauf des E-Books in unserem Verlagsshop ist Ihr E-Book DRM-frei.

Viele Grüße und viel Spaß beim Lesen,

*Ihr mitp-Verlagsteam*



Neuerscheinungen, Praxistipps, Gratiskapitel,  
Einblicke in den Verlagsalltag –  
gibt es alles bei uns auf Instagram und Facebook



[instagram.com/mitp\\_verlag](https://instagram.com/mitp_verlag)



[facebook.com/mitp.verlag](https://facebook.com/mitp.verlag)

# Inhaltsverzeichnis

## Impressum

## Einleitung

- Vorkenntnisse
- Aufbau des Buches
- Benötigte Software
- Downloads

## Autorin

## Kapitel 1: Klassendefinition und Objektinstanziierung

### 1.1 Klassen und Objekte

- ★ Aufgabe 1.1: Definition einer Klasse
- ★ Aufgabe 1.2: Objekt (Instanz) einer Klasse erzeugen

### 1.2 Das Überladen von Methoden

- ★ Aufgabe 1.3: Eine Methode überladen

### 1.3 Die Datenkapselung, ein Prinzip der objektorientierten Programmierung

- ★ Aufgabe 1.4: Zugriffsmethoden

### 1.4 Das »aktuelle Objekt« und die »this-Referenz«

- ★★ Aufgabe 1.5: Konstruktordefinitionen

### 1.5 Die Wert- und Referenzübergabe in Methodenaufrufen

- ★★★ Aufgabe 1.6: Wertübergabe in Methoden (»call by value«)

## 1.6 Globale und lokale Referenzen

★★ Aufgabe 1.7: Der Umgang mit Referenzen

## 1.7 Java-Pakete

★ Aufgabe 1.8: Die package-Anweisung

★ Aufgabe 1.9: Die import-Anweisung

## 1.8 Die Modifikatoren für Felder und Methoden in Zusammenhang mit der Definition von Paketen

★★ Aufgabe 1.10: Pakete und die Sichtbarkeit von Membern einer Klasse

## 1.9 Standard-Klassen von Java

### 1.10 Die Wrapper-Klassen von Java und das Auto(un)boxing

★★ Aufgabe 1.11: Das Auto(un)boxing

### 1.11 Das Paket `java.lang.reflect`

### 1.12 Arrays (Reihungen) und die Klassen `Array` und `Arrays`

★★ Aufgabe 1.12: Der Umgang mit Array-Objekten

### 1.13 Zeichenketten und die Klasse `String`

★ Aufgabe 1.13: Der Umgang mit String-Objekten

★★ Aufgabe 1.14: Der Umgang mit Textblöcken

### 1.14 Sprachmerkmale, die in den weiteren Beispielen genutzt werden

★ Aufgabe 1.15: Methoden mit variablen Argumentenlisten

### 1.15 Den Zugriff auf Klassen und Felder minimieren: Unveränderliche (immutable) Klassen und Objekte

## 1.16 Die alte und neue Date&Time-API als Beispiel für veränderliche und unveränderliche Klassen

★★★ Aufgabe 1.16: Die Methoden von Date/Time-Klassen

## 1.17 Private Konstruktoren

★ Aufgabe 1.17: Objekte mithilfe eines privaten Konstruktors erzeugen

## 1.18 Lösungen

[Lösung 1.1](#)

[Lösung 1.2](#)

[Lösung 1.3](#)

[Lösung 1.4](#)

[Lösung 1.5](#)

[Lösung 1.6](#)

[Lösung 1.7](#)

[Lösung 1.8](#)

[Lösung 1.9](#)

[Lösung 1.10](#)

[Lösung 1.11](#)

[Lösung 1.12](#)

[Lösung 1.13](#)

[Lösung 1.14](#)

[Lösung 1.15](#)

[Lösung 1.16](#)

[Lösung 1.17](#)

# Kapitel 2: Abgeleitete Klassen und Vererbung

## 2.1 Abgeleitete Klassen

## 2.2 Die Konstruktoren von abgeleiteten Klassen

## 2.3 Abgeleitete Klassen und die Sichtbarkeit von Feldern und Methoden

★★ Aufgabe 2.1: Test von Sichtbarkeitsebenen im Zusammenhang mit abgeleiteten Klassen

## 2.4 Das Verdecken von Klassenmethoden und das statische Binden von Methoden

★★ Aufgabe 2.2: Der Aufruf von verdeckten Klassenmethoden

## 2.5 Das Überschreiben von Instanzmethoden und das dynamische Binden von Methoden

★ Aufgabe 2.3: Das dynamische Binden von Methoden

## 2.6 Vererbung und Komposition

★ Aufgabe 2.4: Die Komposition

★ Aufgabe 2.5: Die Vererbung

## 2.7 Kovariante Rückgabetypen in Methoden

★★ Aufgabe 2.6: Die Benutzung von kovarianten Rückgabetypen

## 2.8 Verdeckte Felder

## 2.9 Vergrößernde und verkleinernde Konvertierung (»up- und down-casting«)

## 2.10 Der Polymorphismus, ein Prinzip der objektorientierten Programmierung

★★ Aufgabe 2.7: Der »Subtyp-Polymorphismus« im Kontext einer Klassenhierarchie

## 2.11 Die Methoden der Klassen `java.lang.Object` und `java.util.Objects`

★ Aufgabe 2.8: Die `equals()`- und `hashCode()`-Methoden von `Object`

★★★ Aufgabe 2.9: Die equals()-Methode und die Vererbung

2.12 Das Klonen und die Gleichheit von geklonten Objekten

★ Aufgabe 2.10: Das Klonen von Instanzen der eigenen Klasse

★ Aufgabe 2.11: Das Klonen von Instanzen anderer Klassen

★ Aufgabe 2.12: Das Klonen und der Copy-Konstruktor

2.13 Der Garbage Collector und das Beseitigen von Objekten

2.14 Lösungen

[Lösung 2.1](#)

[Lösung 2.2](#)

[Lösung 2.3](#)

[Lösung 2.4](#)

[Lösung 2.5](#)

[Lösung 2.6](#)

[Lösung 2.7](#)

[Lösung 2.8](#)

[Lösung 2.9](#)

[Lösung 2.10](#)

[Lösung 2.11](#)

[Lösung 2.12](#)

**Kapitel 3: Die Definition von abstrakten Klassen, Interfaces und Annotationen**

3.1 Abstrakte Klassen

## 3.2 Abstrakte Java-Standard-Klassen und eigene Definitionen von abstrakten Klassen

★ Aufgabe 3.1: Die abstrakte Klasse Number und ihre Unterklassen

★ Aufgabe 3.2: Definition einer eigenen abstrakten Klasse

## 3.3 Interfaces (Schnittstellen)

★★ Aufgabe 3.3: Die Definition eines Interface

## 3.4 Die Entscheidung zwischen abstrakten Klassen und Interfaces

★★★ Aufgabe 3.4: Paralleler Einsatz von Interfaces und abstrakten Klassen

## 3.5 Implementieren mehrerer Interfaces für eine Klasse

★★ Aufgabe 3.5: Das Ableiten von Interfaces

## 3.6 Die Definition von inneren Klassen

★ Aufgabe 3.6: Ein Beispiel mit anonymer Klasse

## 3.7 Erweiterungen in der Definition von Interfaces

★★★ Aufgabe 3.7: Private Interface-Methoden

## 3.8 Die Definition von Annotationen

### 3.9 Vordefinierte Annotationstypen

★ Aufgabe 3.8: Annotationen an Methoden und Parameter von Methoden anheften

★ Aufgabe 3.9: Eine Klasse annotieren

★★ Aufgabe 3.10: Die @Override- und @Inherited-Annotation

## 3.10 Lösungen

Lösung 3.1

Lösung 3.2

[Lösung 3.3](#)  
[Lösung 3.4](#)  
[Lösung 3.5](#)  
[Lösung 3.6](#)  
[Lösung 3.7](#)  
[Lösung 3.8](#)  
[Lösung 3.9](#)  
[Lösung 3.10](#)

## Kapitel 4: Generics

[4.1 Die Generizität](#)

[4.2 Generische Klassen und Interfaces](#)

 [Aufgabe 4.1: Generischer Datentyp als Behälter für die Instanzen vom Typ des Klassenparameters](#)

 [Aufgabe 4.2: Generischer Datentyp als »Über-Typ« für die Instanzen vom Typ des Klassenparameters](#)

[4.3 Wildcardtypen](#)

 [Aufgabe 4.3: Ungebundene Wildcardtypen](#)

 [Aufgabe 4.4: Obere und untere Schranken für Wildcardtypen](#)

[4.4 Legacy Code, Erasure und Raw-Typen](#)

 [Aufgabe 4.5: Raw-Typen am Beispiel einer generischen Klasse mit zwei Typparametern](#)

 [Aufgabe 4.6: Brückenmethoden \(»bridge methods«\)](#)

[4.5 Generische Arrays](#)

 [Aufgabe 4.7: Erzeugen von generischen Arrays](#)

## 4.6 Generische Methoden

★★ Aufgabe 4.8: Generische Methodendefinitionen

## 4.7 Generische Standard-Klassen und -Interfaces

### 4.8 for-each-Schleifen für Collections

★ Aufgabe 4.9: Das Interface List<E> und die Klasse ArrayList<E>

★★ Aufgabe 4.10: Das Interface Collection<E> und die Klasse Vector<E>

★★ Aufgabe 4.11: Das Interface Map<K,V> und die Klasse TreeMap<K,V>

### 4.9 Factory-Methoden in Collections

★ Aufgabe 4.12: Factory-Methoden für List, Set und Map

## 4.10 Die Interfaces Enumeration<E>, Iterable<T> und Iterator<E>

## 4.11 Enumerationen und die generische Klasse Enum<E extends Enum<E>>

★★ Aufgabe 4.13: Die Definition von Enumerationen

## 4.12 Die Interfaces Comparable<T> und Comparator<T> und das Sortieren von Objekten

★★★ Aufgabe 4.14: Das Comparable<T>-Interface

★★★ Aufgabe 4.15: Comparable<T> versus Comparator<T>

## 4.13 Typinferenz für Methoden

## 4.14 Typinferenz beim Erzeugen von Instanzen eines generischen Typs

★★ Aufgabe 4.16: Typinferenz beim Instanziieren von generischen Klassen



## Aufgabe 4.17: Der Diamond-Operator in Java

9

### 4.15 Lösungen

[Lösung 4.1](#)

[Lösung 4.2](#)

[Lösung 4.3](#)

[Lösung 4.4](#)

[Lösung 4.5](#)

[Lösung 4.6](#)

[Lösung 4.7](#)

[Lösung 4.8](#)

[Lösung 4.9](#)

[Lösung 4.10](#)

[Lösung 4.11](#)

[Lösung 4.12](#)

[Lösung 4.13](#)

[Lösung 4.14](#)

[Lösung 4.15](#)

[Lösung 4.16](#)

[Lösung 4.17](#)

## Kapitel 5: Exceptions und Errors

[5.1 Ausnahmen auslösen](#)

[5.2 Ausnahmen abfangen oder weitergeben](#)



[Aufgabe 5.1: Unbehandelte](#)

[RuntimeExceptions](#)



[Aufgabe 5.2: Behandelte RuntimeExceptions](#)

[5.3 Das Verwenden von finally in der Ausnahmebehandlung](#)



[Aufgabe 5.3: Der finally-Block](#)

## 5.4 Ausnahmen manuell auslösen

## 5.5 Exception-Unterklassen erzeugen

★ Aufgabe 5.4: Benutzerdefinierte Ausnahmen manuell auslösen

## 5.6 Multi-catch-Klausel und verbesserte Typprüfung beim Rethrowing von Exceptions

★★ Aufgabe 5.5: Disjunction-Typ für Exceptions

★★ Aufgabe 5.6: Typprüfung beim Rethrowing von Exceptions

## 5.7 Lösungen

Lösung 5.1

Lösung 5.2

Lösung 5.3

Lösung 5.4

Lösung 5.5

Lösung 5.6

# Kapitel 6: Lambdas und Streams

## 6.1 Mittels anonymer Klassen Code an Methoden übergeben

## 6.2 Funktionale Interfaces

## 6.3 Syntax und Deklaration von Lambda-Ausdrücken

★ Aufgabe 6.1: Lambda-Ausdruck ohne Parameter versus anonymer Klasse

★ Aufgabe 6.2: Lambda-Ausdruck mit Parameter versus anonymer Klasse

## 6.4 Scoping und Variable Capture

★★ Aufgabe 6.3: Die Umgebung von Lambda-Ausdrücken

## 6.5 Methoden- und Konstruktor-Referenzen

★ Aufgabe 6.4: Methoden-Referenzen in Zuweisungen

★★ Aufgabe 6.5: Konstruktor-Referenzen und die neuen funktionalen Interfaces Supplier<T> und Function<T,R>

6.6 Default- und statische Methoden in Interfaces

6.7 Das neue Interface Stream

6.8 Die forEach-Methoden von Iterator, Iterable und Stream

★ Aufgabe 6.6: Die funktionalen Interfaces BiConsumer<T,U>, BiPredicate<T,U> und BiFunction<T,U,R>

★★ Aufgabe 6.7: Die Methoden des Interface Stream und die Behandlung von Exceptions in Lambda-Ausdrücken

6.9 Das Interface Collector und die Klasse Collectors: Reduktion mittels Methoden von Streams und Kollektoren.

★★ Aufgabe 6.8: Weitere Methoden des Interface Stream: limit(), count(), max(), min(), skip(), reduce() und collect()

★★★ Aufgabe 6.9: Das Interface Collector und die Klasse Collectors

6.10 Parallele Streams

★★★ Aufgabe 6.10: Parallele Streams

6.11 Die map()- und flatMap()-Methoden von Stream und Optional

★★ Aufgabe 6.11: map() versus flatMap()

6.12 Spracherweiterungen mit Java 10, Java 11, Java 12 und Java 13

★★★ Aufgabe 6.12: Typinferenz für lokale Variablen in Java 10 und Java 11

★★★ Aufgabe 6.13: Switch-Statements und Switch-Expressions

## 6.13 Lösungen

[Lösung 6.1](#)

[Lösung 6.2](#)

[Lösung 6.3](#)

[Lösung 6.4](#)

[Lösung 6.5](#)

[Lösung 6.6](#)

[Lösung 6.7](#)

[Lösung 6.8](#)

[Lösung 6.9](#)

[Lösung 6.10](#)

[Lösung 6.11](#)

[Lösung 6.12](#)

[Lösung 6.13](#)

# Kapitel 7: Die Modularität von Java

## 7.1 Das Java-Modulsystem

★ Aufgabe 7.1: Eine einfache Modul-Definition

★★★ Aufgabe 7.2: Eine Applikation mit mehreren Modulen

★★★ Aufgabe 7.3: Implizites Lesen von Modulen

★★★ Aufgabe 7.4: Eine modulbasierte Service-Implementierung

## 7.2 Lösungen

[Lösung 7.1](#)

[Lösung 7.2](#)

[Lösung 7.3](#)

[Lösung 7.4](#)

## **Kapitel 8: Records, Sealed Classes und Pattern Matching**

[8.1 Das Pattern Matching für den instanceof-Operator](#)

[8.2 Der neue Java-Typ Record](#)

[8.3 Sealed Classes in Java](#)

[8.4 Das Pattern Matching für switch](#)

★ Aufgabe 8.1: Die Definition von record-Klassen und das Pattern Matching für den instanceof-Operator

★ Aufgabe 8.2: sealed-, final- und non-sealed-Klassen

★ Aufgabe 8.3: sealed-Interfaces und das Pattern Matching

★★ Aufgabe 8.4: Algebraische Datentypen (ADTs), ein weiterer Schritt in Richtung funktionale Programmierung

★★ Aufgabe 8.5: Das Pattern Matching für switch

[8.5 Lösungen](#)

[Lösung 8.1](#)

[Lösung 8.2](#)

[Lösung 8.3](#)

## **Kapitel 9: JUnit-Tests**

[9.1 JUnit 5 im Überblick](#)

[9.2 Tests schreiben](#)

[9.3 Testen mit dem ConsoleLauncher und der JupiterEngine](#)

- ★ Aufgabe 9.1: Die Klassen App und AppTest
- ★ Aufgabe 9.2: Die Klasse PublishingBookmitOrderingTest
- ★ Aufgabe 9.3: Die Klassen AdditionmitMap und AdditionmitMapTest
- ★★ Aufgabe 9.4: Die Klassen MyClassTest und BuchmitEqualsTest
- ★★ Aufgabe 9.5: Die Klasse TestBeispiele
- ★★ Aufgabe 9.6: Die Klassen RechenOperationenTest und RechenOperationenParametrisierteTests
- ★★ Aufgabe 9.7: Die Klasse AssertThrowsTest

#### 9.4 JUnit-Tests mit Gradle

#### 9.5 Lösungen

[Lösung 9.1](#)

[Lösung 9.3](#)

[Lösung 9.4](#)

[Lösung 9.5](#)

Elisabeth Jung

# **Java Übungsbuch**

**Für die Versionen Java 8 bis Java 17**

**Aufgaben mit vollständigen Lösungen**



# Impressum

## 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 978-3-7475-0451-2

1. Auflage 2021

[www.mitp.de](http://www.mitp.de)

E-Mail: [mitp-verlag@sigloch.de](mailto:mitp-verlag@sigloch.de)

Telefon: +49 7953 / 7189 - 079

Telefax: +49 7953 / 7189 - 082

© 2021 mitp Verlags GmbH & Co. KG

Dieses Werk, einschließlich aller seiner Teile, ist urheberrechtlich geschützt. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtsgesetzes ist ohne Zustimmung des Verlages unzulässig und strafbar. Dies gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen und die Einspeicherung und Verarbeitung in elektronischen Systemen.

Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften.

Lektorat: Sabine Schulz

Sprachkorrektorat: Petra Heubach-Erdmann

Covergestaltung: © maxec / [stock.adobe.com](http://stock.adobe.com)

electronic **public**ation: III-satz, Husby, [www.drei-satz.de](http://www.drei-satz.de)

Dieses Ebook verwendet das ePub-Format und ist optimiert für die Nutzung mit dem iBooks-reader auf dem iPad von Apple. Bei der Verwendung anderer Reader kann es zu Darstellungsproblemen kommen.

Der Verlag räumt Ihnen mit dem Kauf des ebooks das Recht ein, die Inhalte im Rahmen des geltenden Urheberrechts zu nutzen. Dieses Werk, einschließlich aller seiner Teile, ist urheberrechtlich geschützt. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtsgesetzes ist ohne Zustimmung des Verlages unzulässig und strafbar. Dies gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen und Einspeicherung und Verarbeitung in elektronischen Systemen.

Der Verlag schützt seine ebooks vor Missbrauch des Urheberrechts durch ein digitales Rechtemanagement. Bei Kauf im Webshop des Verlages werden die ebooks mit einem nicht sichtbaren digitalen Wasserzeichen individuell pro Nutzer signiert.

Bei Kauf in anderen ebook-Webshops erfolgt die Signatur durch die Shopbetreiber. Angaben zu diesem DRM finden Sie auf den Seiten der jeweiligen Anbieter.

# Einleitung

*Das Java Übungsbuch: Für die Versionen Java 8 bis Java 17* ist wie alle meine Übungsbücher aus der Erkenntnis entstanden, dass zu umfangreiche Beispiele mit komplizierten Algorithmen beim Lernen von Java am Anfang keine echte Hilfe bieten. Darum liegt der Schwerpunkt des Buches nicht auf der Umsetzung von komplizierten Vorgängen, sondern konzentriert sich stattdessen darauf, die in der Dokumentation nicht immer verständlich formulierten Erläuterungen zu Java-Klassen und -Interfaces mit einfachen Beispielen zu erklären und gleichzeitig die zugrunde liegenden Konzepte zu erörtern.

*Das Java Übungsbuch: Für die Versionen Java 8 bis Java 17* wendet sich in erster Linie an Lehrer, Schüler und Studenten als Begleitliteratur zum Lernen der Programmiersprache Java, ist aber auch zum Selbststudium für alle Interessenten an dem Erlernen der Programmiersprache geeignet.

Durch die Einfachheit und Vollständigkeit der Aufgabenlösungen sowie die unterschiedlichen Lösungsmöglichkeiten erhält der Leser ein fundiertes Verständnis für die Aufgabenstellungen und deren Lösungen.

Durch das Lösen von Aufgaben soll der in Referenz- und Lehrbüchern von Java angebotene Stoff vertieft werden, und die dabei erzielten Ergebnisse können anhand der Lösungsvorschläge überprüft werden. Die Beispiele im Buch sind eher selten von zu komplexer Natur, sodass der eigentliche Zweck nicht in den Hintergrund tritt, und alle beschriebenen Themen können tiefgehend und präzise damit eingeübt werden.

# **Vorkenntnisse**

Es ist Voraussetzung, dass der Leser zusätzlich mit einem Lehrbuch zu Java arbeitet bzw. bereits damit gearbeitet hat. Die grundlegenden Erläuterungen zu Java in diesem Buch können lediglich als Wiederholung des bereits vorhandenen Wissens dienen, reichen aber nicht aus, um die Sprache Java erst neu zu lernen.

Als weitere Voraussetzung gelten Grundlagen im Bereich der Programmierung und im Umgang mit dem Betriebssystem. Ein paralleler Zugriff auf die Java-Online-Dokumentation kann Hilfe zu den Java-Standard-Klassen bieten.

# **Aufbau des Buches**

Jedes Kapitel beginnt mit einer kurzen und knappen Wiederholung des Stoffes, der in den Übungsaufgaben dieses Kapitels verwendet wird. Danach folgen alle Aufgabenstellungen der Übungen. Am Ende des Kapitels stehen gesammelt die Lösungen der Übungsaufgaben mit Kommentaren, Erläuterungen und Hinweisen.

Die Aufgaben haben unterschiedliche Schwierigkeitsgrade. Dieser wird im Aufgabekopf durch ein bis drei Sternchen gekennzeichnet:



ein Sternchen für besonders einfache Aufgaben, die auch von Anfängern leicht bewältigt werden können



zwei Sternchen für etwas kompliziertere Aufgaben, die einen durchschnittlichen Aufwand benötigen



drei Sternchen für Aufgaben, die sich an geübte Programmierer richten und einen wesentlich höheren Aufwand oder die Kenntnis von speziellen Details erfordern

Die Programme aus früheren Übungen werden teilweise in späteren Übungen gebraucht und es wird auch immer wieder auf theoretische Zusammenhänge zurückgekommen oder hingewiesen.

Die Lösungsvorschläge haben umfangreiche Kommentare, sodass ein Verständnis für die durchgeführte Aufgabe auch daraus abgeleitet werden kann und dadurch jede einzelne Aufgabe im Gesamtkontext unabhängig erscheint.

In den Kapiteln [1](#), [2](#) und [3](#) liegt das Hauptmerkmal auf den Eigenheiten der objektorientierten Programmierung mit Java. Durch eine Vielzahl von Beispielen wird gezeigt, was die Java-Standard-Klassen und Interfaces an Funktionalitäten bieten und wie diese sinnvoll in die Definition von eigenen Klassen eingebettet werden können. Diese Kapitel enthalten zusätzlich Informationen zur Reflection-API von Java, der Definition von Annotationen und inneren Klassen sowie Neuerungen aus den Versionen 8 bis 13, die sich auf die neue Date&Time-API, Textblöcke, Compact Strings und die Weiterentwicklung von Interfaces beziehen. Mit Java 8 wurden sogenannte Default-Methoden eingeführt. Diese werden in der Literatur auch als »virtual extension«- bzw. »defender«-Methoden bezeichnet und

Schnittstellen, die über derartige Methoden verfügen, als erweiterte Schnittstellen. Damit können Interfaces zusätzlich zu abstrakten Methoden konkrete Methoden in Form von Standard-Implementierungen definieren und in Java wird die Mehrfachvererbung von Funktionalität ermöglicht. Neben Default-Methoden können Interfaces in Java nun auch statische Methoden enthalten. Anders als die statischen Methoden von Klassen werden diese jedoch nicht von abgeleiteten Typen geerbt.

[Kapitel 4](#) beschäftigt sich im Detail mit Generics und dem Collection Framework mit all seinen generischen Klassen und Interfaces sowie mit der Definition von Enumerationen. Die Typinferenz für Methoden und beim Erzeugen von generischen Typen (der Diamond-Operator) sowie das Subtyping von parametrisierten und Wildcard-parametrisierten Typen sind ebenfalls Gegenstand der Themen aus diesem Kapitel.

[Kapitel 5](#) erläutert das Exception-Handling.

[Kapitel 6](#) beschäftigt sich mit den neuen Sprachmitteln von Java 8, Lambdas und Streams sowie mit weiteren Neuerungen aus den Versionen 8 bis 14, wie Switch-Expressions und Local Variable Type Inference.

Mit der Java-Version 8 haben sich ganz neue Betrachtungsweisen und Programmietechniken in der Entwicklung von Applikationen mit Java eröffnet. Eine der wichtigsten Neuerungen in Java 8 sind neue Sprachmittel, die sogenannten Lambda-Ausdrücke, eine Art anonyme Methoden, die auf funktionalen Interfaces basieren. Diese besitzen jedoch eine viel kompaktere Syntax als Methoden. Das resultiert daraus, dass in ihrer Benutzung auf Namen, Modifikatoren, Rückgabetyp, throws-Klausel und in vielen Fällen auch auf Parameter verzichtet werden kann. Mit ihnen

kann Funktionalität ausgeführt, gespeichert und übergeben werden, wie dies bisher nur von Instanzen in Java bekannt war.

Damit verbundene Themen wie die Gegenüberstellung zu anonymen Klassen, Syntax und Semantik, Behandlung von Exceptions, Scoping und Variable Capture, Methoden- und Konstruktor-Referenzen werden in den ersten Unterkapiteln des 6. Kapitels dieses Buches beschrieben und anhand von vielen Beispielen erläutert.

Des Weiteren finden Sie hier die Beschreibung aller neuen funktionalen Interfaces und deren Methoden. Die nachfolgenden Unterkapitel beschäftigen sich im Detail mit der Definition und Nutzung von Streams. Ein Stream besteht aus einer Folge von Werten (in der Literatur wird auch von Sequenzen von Elementen gesprochen), die nur teilweise von mehreren in einer Pipeline dazwischenliegenden Operationen ausgewertet und durch eine abschließende Operation bereitgestellt werden. Diese Operationen werden in Java als Methodenaufrufe formuliert, die Funktionalität in Form von Lambdas und Methoden-Referenzen entgegennehmen können und diese auf alle Elemente der Folge anwenden.

Mit einer Vielzahl von Aufgaben basierend auf Lambdas, Streams und Kollektoren (in denen Stream-Elemente angesammelt und reduziert werden können) werden die neuen Techniken angewandt und alle neuen Begriffe erklärt.

[Kapitel 7](#) präsentiert das neue Java-Modulsystem. Mit dem neuen Modulsystem wurde Java selbst modular gemacht und es können eigene Applikationen und Bibliotheken modularisiert werden.

Java 9 führt das Modul als eine neue Programmkomponente ein. Das Erzeugen von Modulen und deren Abhängigkeiten führen dazu, dass der Zugriffsschutz in Java 9 restriktiver ist. Das Anlegen der erforderlichen Verzeichnisstrukturen für modulbasierte Applikationen, das Packaging von Modul-Code sowie die Implementierung von Services werden ebenfalls im Detail erklärt. Eine Vielzahl von Applikationen mit ausführlichen .cmd-Dateien für deren Ausführung ergänzen die theoretischen Erläuterungen aus diesem Kapitel.

In [Kapitel 8](#) werden die Weiterentwicklungen aus der Programmiersprache mit den Versionen 14 bis 17 erläutert. Dazu gehören die Einführung von Records und Sealed Classes sowie das Pattern Matching.

Records wurden in der Version 14 entworfen, um die Wiederholungen von repetitivem Code in Datenklassen zu unterdrücken. Sie überlassen dem Compiler eine korrekte Generierung der Methoden `equals()`, `hashCode()`, `toString()` (die in Klassen, um eine Wertegleichheit von Objekten zu ermöglichen, überschrieben werden müssen) und von Zugriffsmethoden.

record-Klassen helfen in Kombination mit den in Java 15 neu eingeführten sealed-Klassen und -Interfaces, die auch mit Java 16 im Preview-Status bleiben und mit Java 17 finalisiert werden, die funktionalen Features von Java zu erweitern, insbesondere das Pattern Matching und in naher Zukunft die Destrukturierung von Objekten.

Sealed Classes und Interfaces sind Java-Datentypen, für die die Definition von Subtypen reduziert wird. Sie können nur von den in ihrer Deklaration angegebenen Typen erweitert bzw. implementiert werden.

Auch wenn es keine direkten Abhängigkeiten zwischen den Previews aus den JEPs 395, 394, 409, 406 und 405 gibt, die die Einführung dieser neuen Java-Datentypen sowie das Pattern Matching in Java beschreiben, so sind die von diesen vorgeschlagenen neuen Java-Features, wie mit vielen Beispielen in den Kapiteln [8](#) und [9](#) illustriert wird, sehr gut zusammen einsetzbar und im weitesten Sinne auch dafür gedacht.

Das Pattern Matching wurde in Java ursprünglich für den Abgleich von regulären Ausdrücken mit einem Text eingesetzt und für einen Vergleich von Typen im Zusammenhang mit dem `instanceof`-Operator und `switch` weiterentwickelt.

Der `instanceof`-Operator wurde erweitert, sodass anstelle eines Typ-Tests ein Musterabgleich-Test (»type test pattern«) durchgeführt wird. Dieser prüft die Übereinstimmung eines Zielobjekts mit einem vorgegebenen Mustertyp und erweist sich als sehr nützlich beim Schreiben von `equals()`-Methoden.

Mit Java 17 sind rund um das Pattern Matching weitere Funktionen im Zusammenhang mit Switch Statements und Switch Expressions realisiert worden. Damit werden die Restriktionen für den Typ des Ausdrucks, der im `switch` übergeben wird, weitestgehend aufgehoben. Bei einem klassischen `switch` waren zugelassen: ganzzahlige primitive Typen (`char`, `byte`, `short`, `int`) und die dazugehörigen Wrapper-Typen (`Character`, `Byte`, `Short`, `Integer`) sowie `String` und enum-Konstanten. Diese Auswahl wurde nun auf ganzzahlige primitive Typen und beliebige Referenztypen erweitert, sodass `class`-, `enum`-, `record`- und `array`-Typen zugelassen sind, die zusammen mit einem `null`-case-Label und einem `default`-Label die Angaben in den `switch-case`-Labels ausmachen können.

Die Destrukturierung von Objekten wird zusammen mit Record Patterns und Array Patterns (JEP 405) die Entwickler von nachfolgenden Java-Versionen weiter beschäftigen.

Neu in dieser Auflage des Buches sind Tests mit JUnit 5 und Gradle, die in [Kapitel 9](#) beispielhaft präsentiert werden.

JUnit 5 kann von der Website <https://junit.org/junit5/> unter »Latest Release« (aktuelle Version zum Zeitpunkt der Redaktion dieses Buches waren: Jupiter v5.7.1, Vintage v5.7.1, Platform v1.7.1) heruntergeladen werden.

Zum Testen von Applikationen werden, wie auch in den bevorstehenden Versionen von JUnit üblich, sogenannte Testklassen geschrieben. Sie beinhalten Methoden, die Testfälle beschreiben, den Rückgabetyp `void` aufweisen und durch Annotationen gekennzeichnet sind.

JUnit 5 führt darüber hinaus das Konzept eines ConsoleLaunchers ein, der benutzt werden kann, um Tests zu entwickeln, zu filtern und durchzuführen.

Um Ihnen ein gutes Verständnis für Details zu ermöglichen, wähle ich in diesem Buch die Ausführung über die Kommandozeile, die der ConsoleLauncher in diesem Fall ermöglicht.

Sicherlich sind Build-Tools wie Gradle und IDEs wie Eclipse, IntelliJ IDEA oder Maven eine große Hilfe nicht nur bei der Ausführung von JUnit-Tests, sondern generell in der Programmierung mit Java. Die Angabe von Details in diesem Zusammenhang würde den Rahmen dieses Buches jedoch sprengen.

In einem Unterkapitel in [Kapitel 9](#) erfolgt eine kurze Beschreibung von Gradle und der Ausführung von Tests mit diesem Tool. Weil es gerade im Zusammenhang mit JUnit-

Tests dem Anwender viel Kopfzerbrechen und Arbeit erspart, präsentiere ich es als Alternative zum ConsoleLauncher für die Durchführung von JUnit-Tests für die Java-Applikationen.

Eine neue Gradle-Version kann von der Website <https://gradle.org/releases/> heruntergeladen werden. Zum Zeitpunkt der Buchredaktion war die Version v7.0.1 aktuell.

Weil der Schwerpunkt des Buches nicht auf der Umsetzung von aufwendigen Algorithmen liegen soll, verwende ich einfache Beispiele mit Zahlen, Buchstaben, Wörtern, Büchern, Wochentagen, geometrischen Figuren etc. und teilweise auch mit ganz abstrakten Klassennamen wie Klasse1, Klasse2, KlasseA, KlasseB etc.

An dieser Stelle möchte ich auf das dem Buch zugrunde liegende Konzept hinweisen, dass parallel zu einfachen Aufgaben, die zu allen eingeführten Definitionen und Begriffen gebracht werden, auch Aufgaben von einem höheren Schwierigkeitsgrad präsentiert werden. Dabei werden anhand von inhaltlichen Zusammenhängen zwischen den Beispielen viele Basiskonzepte von Java erläutert.

Ich habe generell versucht, keine Begriffe, Klassen und Komponenten zu benutzen, die nicht schon in vorangehenden Beispielen und Kapiteln definiert oder erläutert wurden. In den wenigen Fällen, wo es sich nicht vermieden ließ, wird darauf hingewiesen und auf die entsprechenden Stellen verwiesen.

Das Buch soll möglichst parallel zu einer Vielzahl von Java-Lehrbüchern eingesetzt werden können und einen Beitrag dazu leisten, die große Fülle von Informationen, die auf uns über die API-Dokumentation zukommt, besser einzuordnen und korrekt anwenden zu können.