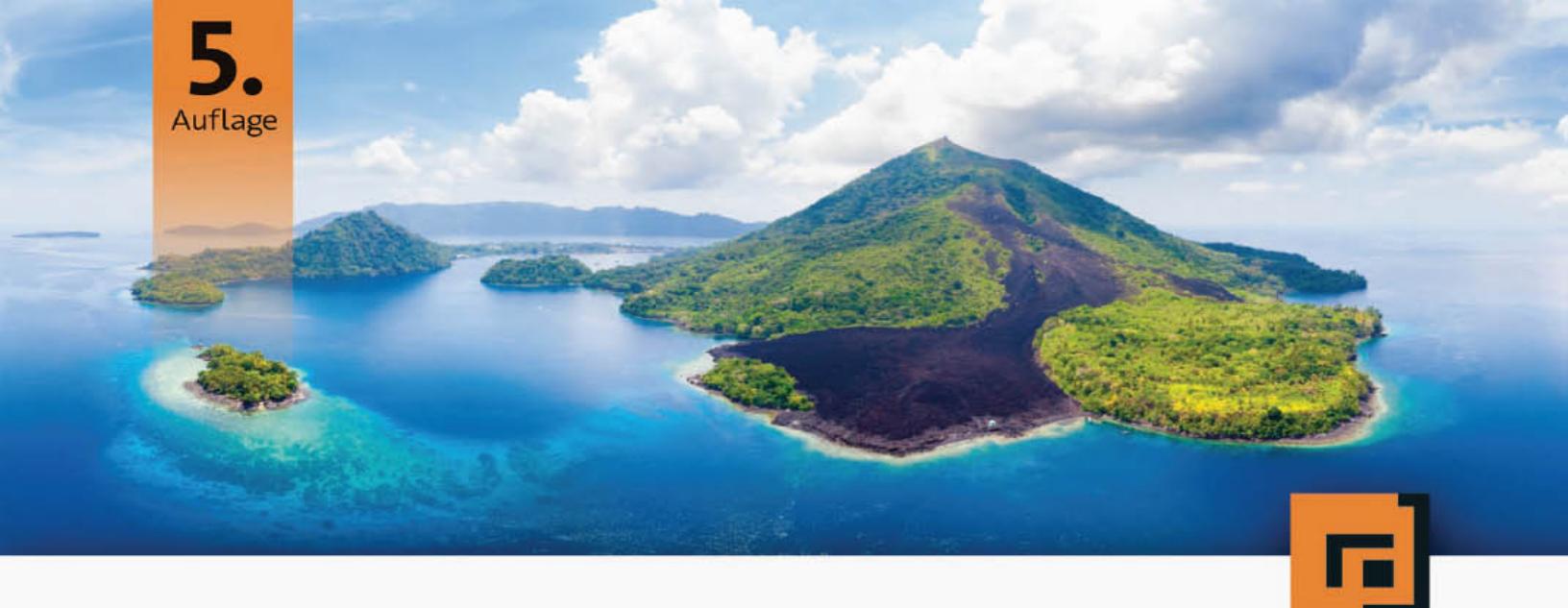


**5.**  
Auflage



Michael Inden

# Der Weg zum Java-Profi

Konzepte und Techniken für die  
professionelle Java-Entwicklung

**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, CTO sowie Leiter Academy gearbeitet. Zurzeit ist er freiberuflich als Autor und Trainer 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 sowie dem Coaching. Sein Wissen gibt er gerne als Trainer in internen und externen Schulungen und auf Konferenzen weiter, etwa bei der JAX/W-JAX, JAX London, Oracle Code One, ch.open sowie bei der Java User Group Switzerland.



Zu diesem Buch – sowie zu vielen weiteren dpunkt.büchern – können Sie auch das entsprechende E-Book im PDF-Format herunterladen. Werden Sie dazu einfach Mitglied bei dpunkt.plus<sup>+</sup>:

**[www.dpunkt.plus](http://www.dpunkt.plus)**

**Michael Inden**

# **Der Weg zum Java-Profi**

**Konzepte und Techniken für die  
professionelle Java-Entwicklung**

5., überarbeitete und aktualisierte Auflage



Michael Inden  
[michael\\_inden@hotmail.com](mailto:michael_inden@hotmail.com)

Lektorat: Dr. Michael Barabas  
Projektkoordinierung/Lektoratsassistenz: Anja Weimer  
Fachgutachten: Torsten Horn, Aachen  
Copy-Editing: Ursula Zimpfer, Herrenberg  
Satz: Michael Inden  
Herstellung: Stefanie Weidner  
Umschlaggestaltung: Helmut Kraus, [www.exclam.de](http://www.exclam.de)

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 978-3-86490-707-4  
PDF 978-3-96088-842-0  
ePub 978-3-96088-843-7  
mobi 978-3-96088-844-4

5., überarbeitete und aktualisierte Auflage 2021

Copyright © 2021 dpunkt.verlag GmbH

Wieblinger Weg 17  
69123 Heidelberg

*Hinweis:*

Der Umwelt zuliebe verzichten wir auf die Einschweißfolie.

*Schreiben Sie uns:*

Falls Sie Anregungen, Wünsche und Kommentare haben, lassen Sie es  
unwissen: [hallo@dpunkt.de](mailto:hallo@dpunkt.de).

Die vorliegende Publikation ist urheberrechtlich geschützt. Alle Rechte vorbehalten. Die Verwendung der Texte und Abbildungen, auch auszugsweise, ist ohne die schriftliche Zustimmung des Verlags urheberrechtswidrig und daher strafbar. Dies gilt insbesondere für die Vervielfältigung, Übersetzung oder die Verwendung in elektronischen Systemen.

Es wird darauf hingewiesen, dass die im Buch verwendeten Soft- und Hardware-Bezeichnungen sowie Markennamen und Produktbezeichnungen der jeweiligen Firmen im Allgemeinen warenzeichen-, marken- oder patentrechtlichem Schutz unterliegen.

Alle Angaben und Programme in diesem Buch wurden mit größter Sorgfalt kontrolliert. Weder Autor noch Verlag können jedoch für Schäden haftbar gemacht werden, die in Zusammenhang mit der Verwendung dieses Buches stehen.

5 4 3 2 1 0

# Inhaltsübersicht

## 1 Einleitung

### I Java-Grundlagen, Analyse und Design

#### 2 Professionelle Arbeitsumgebung

#### 3 Objektorientiertes Design

#### 4 Lambdas, Methodenreferenzen und Defaultmethoden

#### 5 Java-Grundlagen

### II Bausteine stabiler Java-Applikationen

#### 6 Das Collections-Framework

#### 7 Das Stream-API

#### 8 Datumsverarbeitung seit JDK 8

#### 9 Applikationsbausteine

#### 10 Multithreading-Grundlagen

- 11 Modern Concurrency**
- 12 Fortgeschrittene Java-Themen**
- 13 Basiswissen Internationalisierung**

### **III Wichtige Neuerungen in Java 12 bis 15**

- 14 Neues und Änderungen in den Java-Versionen 12 bis 15**

### **IV Modularisierung**

- 15 Modularisierung mit Project Jigsaw**

### **V Fallstricke und Lösungen im Praxisalltag**

- 16 Bad Smells**
- 17 Refactorings**
- 18 Entwurfsmuster**

### **VI Qualitätssicherungsmaßnahmen**

- 19 Programmierstil und Coding Conventions**
- 20 Unit Tests**
- 21 Codereviews**
- 22 Optimierungen**

## **23 Schlussgedanken**

### **VII Anhang**

#### **A Grundlagen zur Java Virtual Machine**

##### **Literaturverzeichnis**

##### **Index**

# Inhaltsverzeichnis

## 1 Einleitung

- 1.1 Über dieses Buch
  - 1.1.1 Motivation
  - 1.1.2 Was leistet dieses Buch und was nicht?
  - 1.1.3 Wie und was soll mithilfe des Buchs gelernt werden?
  - 1.1.4 Wer sollte dieses Buch lesen?
- 1.2 Aufbau des Buchs
- 1.3 Konventionen und ausführbare Programme

## I Java-Grundlagen, Analyse und Design

## 2 Professionelle Arbeitsumgebung

- 2.1 Vorteile von IDEs am Beispiel von Eclipse
- 2.2 Projektorganisation
  - 2.2.1 Projektstruktur in Eclipse
  - 2.2.2 Projektstruktur für Maven und Gradle
- 2.3 Einsatz von Versionsverwaltungen
  - 2.3.1 Arbeiten mit zentralen Versionsverwaltungen
  - 2.3.2 Dezentrale Versionsverwaltungen
  - 2.3.3 VCS und DVCS im Vergleich
- 2.4 Einsatz eines Unit-Test-Frameworks
  - 2.4.1 Das JUnit-Framework

- 2.4.2 Parametrierte Tests mit JUnit 5
- 2.4.3 Vorteile von Unit Tests
- 2.5 Debugging
  - 2.5.1 Fehlersuche mit einem Debugger
  - 2.5.2 Remote Debugging
- 2.6 Deployment von Java-Applikationen
  - 2.6.1 Das JAR-Tool im Kurzüberblick
  - 2.6.2 JAR inspizieren und ändern, Inhalt extrahieren
  - 2.6.3 Metainformationen und das Manifest
  - 2.6.4 Inspizieren einer JAR-Datei
- 2.7 Einsatz eines IDE-unabhängigen Build-Prozesses
  - 2.7.1 Maven im Überblick
  - 2.7.2 Builds mit Gradle
  - 2.7.3 Vorteile von Maven und Gradle
- 2.8 Weiterführende Literatur

### **3 Objektorientiertes Design**

- 3.1 OO-Grundlagen
  - 3.1.1 Grundbegriffe
  - 3.1.2 Beispielentwurf: Ein Zähler
  - 3.1.3 Vom imperativen zum objektorientierten Entwurf
  - 3.1.4 Diskussion der OO-Grundgedanken
  - 3.1.5 Wissenswertes zum Objektzustand
- 3.2 Grundlegende OO-Techniken
  - 3.2.1 Schnittstellen (Interfaces)
  - 3.2.2 Basisklassen und abstrakte Basisklassen
  - 3.2.3 Interfaces und abstrakte Basisklassen
- 3.3 Wissenswertes zu Vererbung
  - 3.3.1 Probleme durch Vererbung
  - 3.3.2 Delegation statt Vererbung
- 3.4 Fortgeschrittenere OO-Techniken
  - 3.4.1 Read-only-Interface

- 3.4.2 Immutable-Klasse
- 3.4.3 Marker-Interface
- 3.4.4 Konstantensammlungen und Aufzählungen
- 3.4.5 Value Object (Data Transfer Object)
- 3.5 Prinzipien guten OO-Designs
  - 3.5.1 Geheimnisprinzip nach Parnas
  - 3.5.2 Law of Demeter
  - 3.5.3 SOLID-Prinzipien
- 3.6 Formen der Varianz
  - 3.6.1 Grundlagen der Varianz
  - 3.6.2 Kovariante Rückgabewerte
- 3.7 Generische Typen (Generics)
  - 3.7.1 Einführung
  - 3.7.2 Generics und Auswirkungen der Type Erasure
- 3.8 Weiterführende Literatur

## **4 Lambdas, Methodenreferenzen und Defaultmethoden**

- 4.1 Einstieg in Lambdas
  - 4.1.1 Syntax von Lambdas
  - 4.1.2 Functional Interfaces und SAM-Typen
  - 4.1.3 Exceptions in Lambdas
- 4.2 Syntaxerweiterungen in Interfaces
  - 4.2.1 Defaultmethoden
  - 4.2.2 Statische Methoden in Interfaces
- 4.3 Methodenreferenzen
- 4.4 Externe vs. interne Iteration
- 4.5 Wichtige Functional Interfaces für Collections
  - 4.5.1 Das Interface Predicate<T>
  - 4.5.2 Das Interface UnaryOperator<T>
- 4.6 Praxiswissen: Definition von Lambdas

## **5 Java-Grundlagen**

- 5.1 Die Klasse Object

- 5.1.1 Die Methode `toString()`
- 5.1.2 Die Methode `equals()`
- 5.2 Primitive Typen und Wrapper-Klassen
  - 5.2.1 Grundlagen
  - 5.2.2 Konvertierung von Werten
  - 5.2.3 Wissenswertes zu Auto-Boxing und Auto-Unboxing
  - 5.2.4 Ausgabe und Verarbeitung von Zahlen
- 5.3 Stringverarbeitung
  - 5.3.1 Die Klasse `String`
  - 5.3.2 Die Klassen `StringBuffer` und `StringBuilder`
  - 5.3.3 Ausgaben mit `format()` und `printf()`
  - 5.3.4 Die Methode `split()` und reguläre Ausdrücke
  - 5.3.5 Optimierung bei Strings in JDK 9
  - 5.3.6 Neue Methoden in der Klasse `String` in JDK 11
- 5.4 Varianten innerer Klassen
- 5.5 Ein- und Ausgabe (I/O)
  - 5.5.1 Dateibehandlung und die Klasse `File`
  - 5.5.2 Ein- und Ausgabestreams im Überblick
  - 5.5.3 Zeichencodierungen bei der Ein- und Ausgabe
  - 5.5.4 Speichern und Laden von Daten und Objekten
  - 5.5.5 Dateiverarbeitung mit dem NIO
  - 5.5.6 Neue Hilfsmethoden in der Klasse `Files` in JDK 11
- 5.6 Fehlerbehandlung
  - 5.6.1 Einstieg in die Fehlerbehandlung
  - 5.6.2 Checked Exceptions und Unchecked Exceptions
  - 5.6.3 Besonderheiten beim Exception Handling
  - 5.6.4 Exception Handling und Ressourcenfreigabe
  - 5.6.5 Assertions
- 5.7 Weitere Neuerungen in JDK 9, 10 und 11
  - 5.7.1 Erweiterung der `@Deprecated`-Annotation in JDK 9

- 5.7.2 Syntaxerweiterung var in JDK 10 und 11
- 5.7.3 Versionsverarbeitung mit JDK 9 und 10
- 5.8 Weiterführende Literatur

## II Bausteine stabiler Java-Applikationen

### 6 Das Collections-Framework

- 6.1 Datenstrukturen und Containerklassen
  - 6.1.1 Wahl einer geeigneten Datenstruktur
  - 6.1.2 Arrays
  - 6.1.3 Das Interface Collection
  - 6.1.4 Das Interface Iterator
  - 6.1.5 Listen und das Interface List
  - 6.1.6 Mengen und das Interface Set
  - 6.1.7 Grundlagen von hashbasierten Containern
  - 6.1.8 Grundlagen automatisch sortierender Container
  - 6.1.9 Die Methoden equals(), hashCode() und compareTo() im Zusammenspiel
  - 6.1.10 Schlüssel-Wert-Abbildungen und das Interface Map
  - 6.1.11 Erweiterungen am Beispiel der Klasse HashMap
  - 6.1.12 Erweiterungen im Interface Map in JDK 8
  - 6.1.13 Collection-Factory-Methoden in JDK 9
  - 6.1.14 Unveränderliche Kopien von Collections mit Java 10
  - 6.1.15 Entscheidungshilfe zur Wahl von Datenstrukturen
- 6.2 Suchen und Sortieren
  - 6.2.1 Suchen
  - 6.2.2 Sortieren von Arrays und Listen
  - 6.2.3 Sortieren mit Komparatoren
  - 6.2.4 Erweiterungen im Interface Comparator mit JDK 8

- 6.3 Utility-Klassen und Hilfsmethoden
  - 6.3.1 Nützliche Hilfsmethoden
  - 6.3.2 Dekorierte `synchronized` und `unmodifiable`
  - 6.3.3 Vordefinierte Algorithmen in der Klasse `Collections`
- 6.4 Containerklassen: Generics und Varianz
- 6.5 Die Klasse `Optional<T>`
  - 6.5.1 Grundlagen zur Klasse `Optional`
  - 6.5.2 Weiterführendes Beispiel und Diskussion
  - 6.5.3 Verkettete Methodenaufrufe
  - 6.5.4 Erweiterungen in der Klasse `Optional<T>` in JDK 9
  - 6.5.5 Erweiterung in `Optional<T>` in JDK 10 und 11
- 6.6 Fallstricke im Collections-Framework
  - 6.6.1 Wissenswertes zu Arrays
  - 6.6.2 Wissenswertes zu Stack, Queue und Deque
- 6.7 Weiterführende Literatur

## 7 Das Stream-API

- 7.1 Grundlagen zu Streams
  - 7.1.1 Streams erzeugen - Create Operations
  - 7.1.2 Intermediate und Terminal Operations im Überblick
  - 7.1.3 Zustandslose Intermediate Operations
  - 7.1.4 Zustandsbehaftete Intermediate Operations
  - 7.1.5 Terminal Operations
  - 7.1.6 Wissenswertes zur Parallelverarbeitung
  - 7.1.7 Neuerungen im Stream-API in JDK 9
  - 7.1.8 Neuerungen im Stream-API in JDK 10
- 7.2 Filter-Map-Reduce
  - 7.2.1 Herkömmliche Realisierung
  - 7.2.2 Filter-Map-Reduce mit JDK 8
- 7.3 Praxisbeispiele

- 7.3.1 Aufbereiten von Gruppierungen und Histogrammen
- 7.3.2 Maps nach Wert sortieren

## **8 Datumsverarbeitung seit JDK 8**

- 8.1 Überblick über die neu eingeführten Typen
  - 8.1.1 Neue Aufzählungen, Klassen und Interfaces
  - 8.1.2 Die Aufzählungen DayOfWeek und Month
  - 8.1.3 Die Klassen MonthDay, YearMonth und Year
  - 8.1.4 Die Klasse Instant
  - 8.1.5 Die Klasse Duration
  - 8.1.6 Die Aufzählung ChronoUnit
  - 8.1.7 Die Klassen LocalDate, LocalTime und LocalDateTime
  - 8.1.8 Die Klasse Period
  - 8.1.9 Die Klasse ZonedDateTime
  - 8.1.10 Zeitzonen und die Klassen ZoneId und ZoneOffset
  - 8.1.11 Die Klasse Clock
  - 8.1.12 Formatierung und Parsing
- 8.2 Datumsarithmetik
  - 8.2.1 Einstieg in die Datumsarithmetik
  - 8.2.2 Real-World-Example: Gehaltszahltag
- 8.3 Interoperabilität mit Legacy-Code

## **9 Applikationsbausteine**

- 9.1 Einsatz von Bibliotheken
- 9.2 Google Guava im Kurzüberblick
  - 9.2.1 String-Aktionen
  - 9.2.2 Stringkonkatenation und -extraktion
  - 9.2.3 Erweiterungen für Collections
  - 9.2.4 Weitere Utility-Funktionalitäten
- 9.3 Wertebereichs- und Parameterprüfungen
- 9.4 Logging-Frameworks

- 9.4.1 Apache log4j2
- 9.4.2 Tipps und Tricks zum Einsatz von Logging mit log4j2
- 9.5 Konfigurationsparameter und -dateien
  - 9.5.1 Einlesen von Kommandozeilenparametern
  - 9.5.2 Verarbeitung von Properties
  - 9.5.3 Weitere Möglichkeiten zur Konfigurationsverwaltung

## **10 Multithreading-Grundlagen**

- 10.1 Threads und Runnables
  - 10.1.1 Definition der auszuführenden Aufgabe
  - 10.1.2 Start, Ausführung und Ende von Threads
  - 10.1.3 Lebenszyklus von Threads und Thread-Zustände
  - 10.1.4 Unterbrechungswünsche durch Aufruf von `interrupt()`
- 10.2 Zusammenarbeit von Threads
  - 10.2.1 Konkurrierende Datenzugriffe
  - 10.2.2 Locks, Monitore und kritische Bereiche
  - 10.2.3 Deadlocks und Starvation
  - 10.2.4 Kritische Bereiche und das Interface Lock
- 10.3 Kommunikation von Threads
  - 10.3.1 Kommunikation mit Synchronisation
  - 10.3.2 Kommunikation über die Methoden `wait()`, `notify()` und `notifyAll()`
  - 10.3.3 Abstimmung von Threads
  - 10.3.4 Unerwartete `IllegalMonitorStateExceptions`
- 10.4 Das Java-Memory-Modell
  - 10.4.1 Sichtbarkeit
  - 10.4.2 Atomarität
  - 10.4.3 Reorderings
- 10.5 Besonderheiten bei Threads
  - 10.5.1 Verschiedene Arten von Threads

- 10.5.2 Exceptions in Threads
- 10.5.3 Sicheres Beenden von Threads

## 10.6 Weiterführende Literatur

# 11 Modern Concurrency

- 11.1 Concurrent Collections
  - 11.1.1 Thread-Sicherheit und Parallelität mit »normalen« Collections
  - 11.1.2 Parallelität mit den Concurrent Collections
  - 11.1.3 Blockierende Warteschlangen und das Interface Blocking-Queue<E>
- 11.2 Das Executor-Framework
  - 11.2.1 Einführung
  - 11.2.2 Definition von Aufgaben
  - 11.2.3 Parallel Abarbeitung im ExecutorService
- 11.3 Das Fork-Join-Framework
  - 11.3.1 Einführendes Beispiel
  - 11.3.2 Real-World-Example: Merge Sort
- 11.4 Die Klasse CompletableFuture<T>
  - 11.4.1 Einführung
  - 11.4.2 Beispiel: Parallel Verarbeitung von Dateiinhalten
  - 11.4.3 Erweiterungen in JDK 9
  - 11.4.4 Beispiel: Von synchron zu multithreaded
- 11.5 Reactive Streams und die Klasse Flow
  - 11.5.1 Schnelleinstieg Reactive Streams
  - 11.5.2 Reactive Streams im JDK
  - 11.5.3 Beispiel zur Klasse Flow
  - 11.5.4 Fazit
- 11.6 Weiterführende Literatur

# 12 Fortgeschrittene Java-Themen

- 12.1 Crashkurs Reflection
  - 12.1.1 Grundlagen

- 12.1.2 Zugriff auf Methoden und Attribute
- 12.1.3 Spezialfälle
- 12.1.4 Type Erasure und Typinformationen bei Generics
- 12.1.5 Fazit
- 12.2 Annotations
  - 12.2.1 Einführung in Annotations
  - 12.2.2 Standard-Annotations des JDKs
  - 12.2.3 Definition eigener Annotations
  - 12.2.4 Annotations zur Laufzeit auslesen
- 12.3 Serialisierung
  - 12.3.1 Grundlagen der Serialisierung
  - 12.3.2 Die Serialisierung anpassen
  - 12.3.3 Versionsverwaltung der Serialisierung
  - 12.3.4 Optimierung der Serialisierung
- 12.4 Garbage Collection
  - 12.4.1 Grundlagen zur Garbage Collection
  - 12.4.2 Der Garbage Collector »G1«
- 12.5 Dynamic Proxies
  - 12.5.1 Statischer Proxy
  - 12.5.2 Dynamischer Proxy
- 12.6 HTTP/2-API
  - 12.6.1 Einführung
  - 12.6.2 Real-World-Example: Wechselkurs mit REST
  - 12.6.3 Fazit
- 12.7 Weiterführende Literatur

## **13 Basiswissen Internationalisierung**

- 13.1 Internationalisierung im Überblick
  - 13.1.1 Grundlagen und Normen
  - 13.1.2 Die Klasse Locale
  - 13.1.3 Die Klasse PropertyResourceBundle
  - 13.1.4 Formatierte Ein- und Ausgabe

- 13.1.5 Datumswerte und die Klasse DateFormat
- 13.1.6 Zahlen und die Klasse NumberFormat
- 13.1.7 Textmeldungen und die Klasse MessageFormat
- 13.1.8 Stringvergleiche mit der Klasse Collator
- 13.2 Programmbausteine zur Internationalisierung
  - 13.2.1 Unterstützung mehrerer Datumsformate
  - 13.2.2 Fazit und Ausblick

### **III Wichtige Neuerungen in Java 12 bis 15**

- 14 Neues und Änderungen in den Java-Versionen 12 bis 15**
  - 14.1 Syntaxneuerungen
    - 14.1.1 Text Blocks
    - 14.1.2 Switch Expressions
    - 14.1.3 Records (Preview)
    - 14.1.4 Pattern Matching bei instanceof (Preview)
    - 14.1.5 Sealed Types (Preview)
    - 14.1.6 Lokale Enums und Interfaces (Preview)
  - 14.2 API-Neuerungen
    - 14.2.1 Neue Methoden in der Klasse String
    - 14.2.2 Neue Hilfsmethode in der Utility-Klasse Files
    - 14.2.3 Der teeing()-Kollektor
  - 14.3 JVM-Neuerungen
    - 14.3.1 Verbesserung bei NullPointerExceptions
    - 14.3.2 Entfernung der JavaScript-Engine
  - 14.4 Microbenchmark Suite
    - 14.4.1 Eigene Microbenchmarks und Varianten davon
    - 14.4.2 Microbenchmarks mit JMH
    - 14.4.3 Fazit zu JMH
  - 14.5 Java 15 – notwendige Anpassungen für Build-Tools und IDEs

- 14.5.1 Java 15 mit Gradle
  - 14.5.2 Java 15 mit Maven
  - 14.5.3 Java 15 mit Eclipse
  - 14.5.4 Java 15 mit IntelliJ
  - 14.5.5 Java 15 mit JShell oder der Kommandozeile
- 14.6 Fazit

## IV Modularisierung

### 15 Modularisierung mit Project Jigsaw

- 15.1 Grundlagen
  - 15.1.1 Begrifflichkeiten
  - 15.1.2 Ziele von Project Jigsaw
- 15.2 Modularisierung im Überblick
  - 15.2.1 Grundlagen zu Project Jigsaw
  - 15.2.2 Beispiel mit zwei Modulen
  - 15.2.3 Packaging
  - 15.2.4 Abhängigkeiten und Modulgraphen
  - 15.2.5 Module des JDKs einbinden
  - 15.2.6 Arten von Modulen
- 15.3 Sichtbarkeiten und Zugriffsschutz
  - 15.3.1 Sichtbarkeiten
  - 15.3.2 Zugriffsschutz und Reflection
- 15.4 Empfehlenswertes Verzeichnislayout für Module
- 15.5 Kompatibilität und Migration
  - 15.5.1 Kompatibilitätsmodus
  - 15.5.2 Migrationsszenarien
  - 15.5.3 Fallstrick bei der Bottom-up-Migration
  - 15.5.4 Beispiel: Migration mit Automatic Modules
  - 15.5.5 Beispiel: Automatic und Unnamed Module
  - 15.5.6 Beispiel: Abwandlung mit zwei Automatic Modules

### 15.5.7 Fazit

## 15.6 Zusammenfassung

## V Fallstricke und Lösungen im Praxisalltag

### 16 Bad Smells

#### 16.1 Programmdesign

- 16.1.1 Bad Smell: Verwenden von Magic Numbers
- 16.1.2 Bad Smell: Konstanten in Interfaces definieren
- 16.1.3 Bad Smell: Zusammengehörende Konstanten nicht als Typ definiert
- 16.1.4 Bad Smell: Casts auf unbekannte Subtypen
- 16.1.5 Bad Smell: Programmcode im Logging-Code
- 16.1.6 Bad Smell: Dominanter Logging-Code
- 16.1.7 Bad Smell: Unvollständige Änderungen nach Copy-Paste
- 16.1.8 Bad Smell: Unvollständige Betrachtung aller Alternativen
- 16.1.9 Bad Smell: Prä-/Postinkrement in komplexeren Statements
- 16.1.10 Bad Smell: Mehrere aufeinanderfolgende Parameter gleichen Typs
- 16.1.11 Bad Smell: Grundloser Einsatz von Reflection
- 16.1.12 Bad Smell: System.exit() mitten im Programm
- 16.1.13 Bad Smell: Variablendeclaration nicht im kleinstmöglichen Sichtbarkeitsbereich

#### 16.2 Klassendesign

- 16.2.1 Bad Smell: Unnötigerweise veränderliche Attribute
- 16.2.2 Bad Smell: Herausgabe von this im Konstruktor
- 16.2.3 Bad Smell: Aufruf abstrakter Methoden im Konstruktor

- 16.2.4 Bad Smell: Mix abstrakter und konkreter Basisklassen
- 16.2.5 Bad Smell: Referenzierung von Subklassen in Basisklassen
- 16.2.6 Bad Smell: Öffentlicher Defaultkonstruktor lediglich zum Zugriff auf Hilfsmethoden
- 16.3 Fehlerbehandlung und Exception Handling
  - 16.3.1 Bad Smell: Unbehandelte Exception
  - 16.3.2 Bad Smell: Unpassender Exception-Typ
  - 16.3.3 Bad Smell: Fangen der allgemeinsten Exception
  - 16.3.4 Bad Smell: Exceptions zur Steuerung des Kontrollflusses
  - 16.3.5 Bad Smell: Unbedachte Rückgabe von null
  - 16.3.6 Bad Smell: Rückgabe von null statt Exception im Fehlerfall
  - 16.3.7 Bad Smell: Sonderbehandlung von Randfällen
  - 16.3.8 Bad Smell: Keine Gültigkeitsprüfung von Eingabeparametern
  - 16.3.9 Bad Smell: Fehlerhafte Fehlerbehandlung
  - 16.3.10 Bad Smell: I/O ohne finally oder ARM
  - 16.3.11 Bad Smell: Resource Leaks durch Exceptions im Konstruktor
- 16.4 Häufige Fallstricke
- 16.5 Weiterführende Literatur

## **17 Refactorings**

- 17.1 Refactorings am Beispiel
- 17.2 Das Standardvorgehen
- 17.3 Kombination von Basis-Refactorings
  - 17.3.1 Refactoring-Beispiel: Ausgangslage und Ziel
  - 17.3.2 Auflösen der Abhängigkeiten
  - 17.3.3 Vereinfachungen
  - 17.3.4 Verlagern von Funktionalität
- 17.4 Der Refactoring-Katalog

- 17.4.1 Reduziere die Sichtbarkeit von Attributen
  - 17.4.2 Minimiere veränderliche Attribute
  - 17.4.3 Reduziere die Sichtbarkeit von Methoden
  - 17.4.4 Ersetze Mutator- durch Business-Methode
  - 17.4.5 Minimiere Zustandsänderungen
  - 17.4.6 Führe ein Interface ein
  - 17.4.7 Spalte ein Interface auf
  - 17.4.8 Führe ein Read-only-Interface ein
  - 17.4.9 Führe ein Read-Write-Interface ein
  - 17.4.10 Lagere Funktionalität in Hilfsmethoden aus
  - 17.4.11 Trenne Informationsbeschaffung und -verarbeitung
  - 17.4.12 Wandle Konstantensammlung in enum um
  - 17.4.13 Entferne Exceptions zur Steuerung des Kontrollflusses
  - 17.4.14 Wandle in Utility-Klasse mit statischen Hilfsmethoden um
  - 17.4.15 Löse if-else / instanceof durch Polymorphie auf
- 17.5 Defensives Programmieren
- 17.5.1 Führe eine Zustandsprüfung ein
  - 17.5.2 Überprüfe Eingabeparameter
- 17.6 Fallstricke bei Refactorings
- 17.7 Weiterführende Literatur

## **18 Entwurfsmuster**

- 18.1 Erzeugungsmuster
- 18.1.1 Erzeugungsmethode
  - 18.1.2 Fabrikmethode (Factory Method)
  - 18.1.3 Erbauer (Builder)
  - 18.1.4 Singleton
  - 18.1.5 Prototyp (Prototype)
- 18.2 Strukturmuster

- 18.2.1 Fassade (Façade)
- 18.2.2 Adapter
- 18.2.3 Dekorierer (Decorator)
- 18.2.4 Kompositum (Composite)
- 18.3 Verhaltensmuster
  - 18.3.1 Iterator
  - 18.3.2 Null-Objekt (Null Object)
  - 18.3.3 Schablonenmethode (Template Method)
  - 18.3.4 Strategie (Strategy)
  - 18.3.5 Befehl (Command)
  - 18.3.6 Proxy
  - 18.3.7 Beobachter (Observer)
  - 18.3.8 MVC-Architektur
- 18.4 Weiterführende Literatur

## VI Qualitätssicherungsmaßnahmen

- ### 19 Programmierstil und Coding Conventions
- 19.1 Grundregeln eines guten Programmierstils
    - 19.1.1 Keep It Human-Readable
    - 19.1.2 Keep It Simple And Short (KISS)
    - 19.1.3 Keep It Natural
    - 19.1.4 Keep It Clean
  - 19.2 Die Psychologie beim Sourcecode-Layout
    - 19.2.1 Gesetz der Ähnlichkeit
    - 19.2.2 Gesetz der Nähe
  - 19.3 Coding Conventions
    - 19.3.1 Grundlegende Namens- und Formatierungsregeln
    - 19.3.2 Namensgebung
    - 19.3.3 Dokumentation
    - 19.3.4 Programmdesign

- 19.3.5 Klassendesign
- 19.3.6 Parameterlisten
- 19.3.7 Logik und Kontrollfluss
- 19.4 Sourcecode-Prüfung mit Tools
  - 19.4.1 Metriken
  - 19.4.2 Sourcecode-Prüfung im Build-Prozess

## **20 Unit Tests**

- 20.1 Testen im Überblick
  - 20.1.1 Was versteht man unter Testen?
  - 20.1.2 Testarten im Überblick
  - 20.1.3 Zuständigkeiten beim Testen
  - 20.1.4 Testen und Qualität
- 20.2 Wissenswertes zu Testfällen
  - 20.2.1 Testfälle mit JUnit definieren
  - 20.2.2 Problem der Kombinatorik beim Bestimmen von Testfällen
- 20.3 Besondere Assertions und Annotations
- 20.4 Parametrierte Tests mit JUnit 5
  - 20.4.1 Einstieg
  - 20.4.2 Verbesserung des Tests der Rabattberechnung
  - 20.4.3 Praxisbeispiel: Berechnung in Testfall vereinfachen
  - 20.4.4 Praxis-Trickkiste
- 20.5 Fortgeschrittene Unit-Test-Techniken
  - 20.5.1 Stellvertreterobjekte / Test-Doubles
  - 20.5.2 Vorarbeiten für das Testen mit Stubs und Mocks
  - 20.5.3 Die Technik EXTRACT AND OVERRIDE
  - 20.5.4 Einstieg in das Testen mit Mocks und Mockito
  - 20.5.5 Abhängigkeiten mit Mockito auflösen
  - 20.5.6 Unit Tests von privaten Methoden
- 20.6 Test Smells

- 20.6.1 Test Smell: Unangebrachtes assertTrue() und assertFalse()
- 20.6.2 Test Smell: Zu viele Asserts im Testfall
- 20.6.3 Test Smell: Asserts ohne Hinweis
- 20.6.4 Test Smell: Einsatz von toString() in assertEquals()
- 20.6.5 Test Smell: Unit Tests zur Prüfung von Laufzeiten
- 20.7 Nützliche Tools für Unit Tests
  - 20.7.1 Hamcrest
  - 20.7.2 AssertJ
  - 20.7.3 MoreUnit
  - 20.7.4 Infinitest
  - 20.7.5 JaCoCo
  - 20.7.6 EclEmma
- 20.8 Umstieg von JUnit 4 auf JUnit 5
  - 20.8.1 Erweiterung im Gradle-Build für JUnit-5-Tests
  - 20.8.2 Veränderungen in den Annotations
  - 20.8.3 Alternativen für JUnit Rules
  - 20.8.4 Veränderungen bei parametrierten Tests
  - 20.8.5 Alternative zur Hamcrest-Integration in JUnit 4
- 20.9 Fazit
- 20.10 Weiterführende Literatur

## **21 Codereviews**

- 21.1 Definition
- 21.2 Probleme und Tipps zur Durchführung
- 21.3 Vorteile von Codereviews
- 21.4 Codereview-Checkliste

## **22 Optimierungen**

- 22.1 Grundlagen
  - 22.1.1 Optimierungsebenen und Einflussfaktoren
  - 22.1.2 Optimierungstechniken

- 22.1.3 CPU-bound-Optimierungsebenen am Beispiel
- 22.1.4 Messungen – Erkennen kritischer Bereiche
- 22.1.5 Abschätzungen mit der O-Notation
- 22.2 Einsatz geeigneter Datenstrukturen
  - 22.2.1 Einfluss von Arrays und Listen
  - 22.2.2 Optimierungen für Set und Map
  - 22.2.3 Design eines Zugriffsinterface
- 22.3 Lazy Initialization
  - 22.3.1 Konsequenzen des Einsatzes der Lazy Initialization
  - 22.3.2 Lazy Initialization mithilfe des PROXY-Musters
- 22.4 Optimierungen am Beispiel
- 22.5 I/O-bound-Optimierungen
  - 22.5.1 Technik – Wahl passender Strategien
  - 22.5.2 Technik – Caching und Pooling
  - 22.5.3 Technik – Vermeidung unnötiger Aktionen
- 22.6 Memory-bound-Optimierungen
  - 22.6.1 Technik – Wahl passender Strategien
  - 22.6.2 Technik – Caching und Pooling
  - 22.6.3 Optimierungen der Stringverarbeitung
  - 22.6.4 Technik – Vermeidung unnötiger Aktionen
- 22.7 CPU-bound-Optimierungen
  - 22.7.1 Technik – Wahl passender Strategien
  - 22.7.2 Technik – Caching und Pooling
  - 22.7.3 Technik – Vermeidung unnötiger Aktionen
- 22.8 Weiterführende Literatur

## **23 Schlussgedanken**

## **VII Anhang**

### **A Grundlagen zur Java Virtual Machine**

- A.1 Wissenswertes rund um die Java Virtual Machine

- A.1.1 Ausführung eines Java-Programms
- A.1.2 Speicherverwaltung und Classloading

## **Literaturverzeichnis**

## **Index**

# **Vorwort**