

Inhaltsübersicht

1	Einleitung	1
I	Java-Grundlagen, Analyse und Design	11
2	Professionelle Arbeitsumgebung	13
3	Objektorientiertes Design	85
4	Lambdas, Methodenreferenzen und Defaultmethoden	199
5	Java-Grundlagen	221
II	Bausteine stabiler Java-Applikationen	325
6	Das Collections-Framework	327
7	Das Stream-API	453
8	Datumsverarbeitung seit JDK 8	497
9	Applikationsbausteine	529
10	Multithreading-Grundlagen	581
11	Modern Concurrency	633
12	Fortgeschrittene Java-Themen	681
13	Basiswissen Internationalisierung	743

III	Wichtige Neuerungen in Java 12 bis 15	771
14	Neues und Änderungen in den Java-Versionen 12 bis 15	773
IV	Modularisierung	823
15	Modularisierung mit Project Jigsaw	825
V	Fallstricke und Lösungen im Praxisalltag	877
16	Bad Smells	879
17	Refactorings	953
18	Entwurfsmuster	1023
VI	Qualitätssicherungsmaßnahmen	1099
19	Programmierstil und Coding Conventions	1101
20	Unit Tests	1143
21	Codereviews	1265
22	Optimierungen	1273
23	Schlussgedanken	1349
VII	Anhang	1351
A	Grundlagen zur Java Virtual Machine	1353
	Literaturverzeichnis	1357
	Index	1361

Inhaltsverzeichnis

1	Einleitung	1
1.1	Über dieses Buch	1
1.1.1	Motivation	1
1.1.2	Was leistet dieses Buch und was nicht?	2
1.1.3	Wie und was soll mithilfe des Buchs gelernt werden?	2
1.1.4	Wer sollte dieses Buch lesen?	4
1.2	Aufbau des Buchs	4
1.3	Konventionen und ausführbare Programme	6

I Java-Grundlagen, Analyse und Design **11**

2	Professionelle Arbeitsumgebung	13
2.1	Vorteile von IDEs am Beispiel von Eclipse	14
2.2	Projektorganisation	16
2.2.1	Projektstruktur in Eclipse	16
2.2.2	Projektstruktur für Maven und Gradle	18
2.3	Einsatz von Versionsverwaltungen	20
2.3.1	Arbeiten mit zentralen Versionsverwaltungen	23
2.3.2	Dezentrale Versionsverwaltungen	28
2.3.3	VCS und DVCS im Vergleich	34
2.4	Einsatz eines Unit-Test-Frameworks	37
2.4.1	Das JUnit-Framework	37
2.4.2	Parametrisierte Tests mit JUnit 5	43
2.4.3	Vorteile von Unit Tests	45
2.5	Debugging	46
2.5.1	Fehlersuche mit einem Debugger	47
2.5.2	Remote Debugging	50
2.6	Deployment von Java-Applikationen	55
2.6.1	Das JAR-Tool im Kurzüberblick	56
2.6.2	JAR inspizieren und ändern, Inhalt extrahieren	57
2.6.3	Metainformationen und das Manifest	58
2.6.4	Inspizieren einer JAR-Datei	60

2.7	Einsatz eines IDE-unabhängigen Build-Prozesses	63
2.7.1	Maven im Überblick	65
2.7.2	Builds mit Gradle	70
2.7.3	Vorteile von Maven und Gradle	82
2.8	Weiterführende Literatur	83
3	Objektorientiertes Design	85
3.1	OO-Grundlagen	86
3.1.1	Grundbegriffe	86
3.1.2	Beispielentwurf: Ein Zähler	98
3.1.3	Vom imperativen zum objektorientierten Entwurf	106
3.1.4	Diskussion der OO-Grundgedanken	111
3.1.5	Wissenswertes zum Objektzustand	114
3.2	Grundlegende OO-Techniken	123
3.2.1	Schnittstellen (Interfaces)	123
3.2.2	Basisklassen und abstrakte Basisklassen	128
3.2.3	Interfaces und abstrakte Basisklassen	130
3.3	Wissenswertes zu Vererbung	132
3.3.1	Probleme durch Vererbung	132
3.3.2	Delegation statt Vererbung	138
3.4	Fortgeschrittenere OO-Techniken	142
3.4.1	Read-only-Interface	142
3.4.2	Immutable-Klasse	148
3.4.3	Marker-Interface	153
3.4.4	Konstantensammlungen und Aufzählungen	154
3.4.5	Value Object (Data Transfer Object)	160
3.5	Prinzipien guten OO-Designs	162
3.5.1	Geheimnisprinzip nach Parnas	163
3.5.2	Law of Demeter	164
3.5.3	SOLID-Prinzipien	167
3.6	Formen der Varianz	178
3.6.1	Grundlagen der Varianz	178
3.6.2	Kovariante Rückgabewerte	182
3.7	Generische Typen (Generics)	184
3.7.1	Einführung	184
3.7.2	Generics und Auswirkungen der Type Erasure	189
3.8	Weiterführende Literatur	197
4	Lambdas, Methodenreferenzen und Defaultmethoden	199
4.1	Einstieg in Lambdas	199
4.1.1	Syntax von Lambdas	199
4.1.2	Functional Interfaces und SAM-Typen	200
4.1.3	Exceptions in Lambdas	204

4.2	Syntaxerweiterungen in Interfaces	208
4.2.1	Defaultmethoden	208
4.2.2	Statische Methoden in Interfaces	211
4.3	Methodenreferenzen	213
4.4	Externe vs. interne Iteration	214
4.5	Wichtige Functional Interfaces für Collections	215
4.5.1	Das Interface <code>Predicate<T></code>	215
4.5.2	Das Interface <code>UnaryOperator<T></code>	218
4.6	Praxiswissen: Definition von Lambdas	219
5	Java-Grundlagen	221
5.1	Die Klasse <code>Object</code>	221
5.1.1	Die Methode <code>toString()</code>	223
5.1.2	Die Methode <code>equals()</code>	227
5.2	Primitive Typen und Wrapper-Klassen	239
5.2.1	Grundlagen	240
5.2.2	Konvertierung von Werten	247
5.2.3	Wissenswertes zu Auto-Boxing und Auto-Unboxing	250
5.2.4	Ausgabe und Verarbeitung von Zahlen	253
5.3	Stringverarbeitung	256
5.3.1	Die Klasse <code>String</code>	257
5.3.2	Die Klassen <code>StringBuffer</code> und <code>StringBuilder</code>	261
5.3.3	Ausgaben mit <code>format()</code> und <code>printf()</code>	264
5.3.4	Die Methode <code>split()</code> und reguläre Ausdrücke	265
5.3.5	Optimierung bei Strings in JDK 9	270
5.3.6	Neue Methoden in der Klasse <code>String</code> in JDK 11	270
5.4	Varianten innerer Klassen	272
5.5	Ein- und Ausgabe (I/O)	276
5.5.1	Dateibehandlung und die Klasse <code>File</code>	276
5.5.2	Ein- und Ausgabestreams im Überblick	280
5.5.3	Zeichencodierungen bei der Ein- und Ausgabe	283
5.5.4	Speichern und Laden von Daten und Objekten	285
5.5.5	Dateiverarbeitung mit dem NIO	293
5.5.6	Neue Hilfsmethoden in der Klasse <code>Files</code> in JDK 11	296
5.6	Fehlerbehandlung	298
5.6.1	Einstieg in die Fehlerbehandlung	298
5.6.2	Checked Exceptions und Unchecked Exceptions	304
5.6.3	Besonderheiten beim Exception Handling	306
5.6.4	Exception Handling und Ressourcenfreigabe	310
5.6.5	Assertions	314

5.7	Weitere Neuerungen in JDK 9, 10 und 11	318
5.7.1	Erweiterung der <code>@Deprecated</code> -Annotation in JDK 9	318
5.7.2	Syntaxerweiterung <code>var</code> in JDK 10 und 11	319
5.7.3	Versionsverarbeitung mit JDK 9 und 10	323
5.8	Weiterführende Literatur	324

II Bausteine stabiler Java-Applikationen 325

6	Das Collections-Framework	327
6.1	Datenstrukturen und Containerklassen	327
6.1.1	Wahl einer geeigneten Datenstruktur	328
6.1.2	Arrays	330
6.1.3	Das Interface <code>Collection</code>	332
6.1.4	Das Interface <code>Iterator</code>	334
6.1.5	Listen und das Interface <code>List</code>	338
6.1.6	Mengen und das Interface <code>Set</code>	346
6.1.7	Grundlagen von hashbasierten Containern	350
6.1.8	Grundlagen automatisch sortierender Container	359
6.1.9	Die Methoden <code>equals()</code> , <code>hashCode()</code> und <code>compareTo()</code> im Zusammenspiel	367
6.1.10	Schlüssel-Wert-Abbildungen und das Interface <code>Map</code>	369
6.1.11	Erweiterungen am Beispiel der Klasse <code>HashMap</code>	378
6.1.12	Erweiterungen im Interface <code>Map</code> in JDK 8	381
6.1.13	Collection-Factory-Methoden in JDK 9	386
6.1.14	Unveränderliche Kopien von Collections mit Java 10	390
6.1.15	Entscheidungshilfe zur Wahl von Datenstrukturen	392
6.2	Suchen und Sortieren	393
6.2.1	Suchen	393
6.2.2	Sortieren von Arrays und Listen	396
6.2.3	Sortieren mit Komparatoren	399
6.2.4	Erweiterungen im Interface <code>Comparator</code> mit JDK 8	401
6.3	Utility-Klassen und Hilfsmethoden	406
6.3.1	Nützliche Hilfsmethoden	406
6.3.2	Dekorierer <code>synchronized</code> und <code>unmodifiable</code>	411
6.3.3	Vordefinierte Algorithmen in der Klasse <code>Collections</code>	415
6.4	Containerklassen: Generics und Varianz	417
6.5	Die Klasse <code>Optional<T></code>	431
6.5.1	Grundlagen zur Klasse <code>Optional</code>	432
6.5.2	Weiterführendes Beispiel und Diskussion	436
6.5.3	Verkettete Methodenaufrufe	438
6.5.4	Erweiterungen in der Klasse <code>Optional<T></code> in JDK 9	439
6.5.5	Erweiterung in <code>Optional<T></code> in JDK 10 und 11	442

6.6	Fallstricke im Collections-Framework	443
6.6.1	Wissenswertes zu Arrays	443
6.6.2	Wissenswertes zu Stack, Queue und Deque	447
6.7	Weiterführende Literatur	450
7	Das Stream-API	453
7.1	Grundlagen zu Streams	453
7.1.1	Streams erzeugen – Create Operations	454
7.1.2	Intermediate und Terminal Operations im Überblick	456
7.1.3	Zustandslose Intermediate Operations	458
7.1.4	Zustandsbehaftete Intermediate Operations	465
7.1.5	Terminal Operations	467
7.1.6	Wissenswertes zur Parallelverarbeitung	476
7.1.7	Neuerungen im Stream-API in JDK 9	481
7.1.8	Neuerungen im Stream-API in JDK 10	484
7.2	Filter-Map-Reduce	485
7.2.1	Herkömmliche Realisierung	486
7.2.2	Filter-Map-Reduce mit JDK 8	488
7.3	Praxisbeispiele	490
7.3.1	Aufbereiten von Gruppierungen und Histogrammen	491
7.3.2	Maps nach Wert sortieren	492
8	Datumsverarbeitung seit JDK 8	497
8.1	Überblick über die neu eingeführten Typen	497
8.1.1	Neue Aufzählungen, Klassen und Interfaces	498
8.1.2	Die Aufzählungen DayOfWeek und Month	500
8.1.3	Die Klassen MonthDay, YearMonth und Year	500
8.1.4	Die Klasse Instant	501
8.1.5	Die Klasse Duration	502
8.1.6	Die Aufzählung ChronoUnit	506
8.1.7	Die Klassen LocalDate, LocalTime und LocalDateTime	507
8.1.8	Die Klasse Period	510
8.1.9	Die Klasse ZonedDateTime	511
8.1.10	Zeitzonen und die Klassen ZoneId und ZoneOffset	512
8.1.11	Die Klasse Clock	514
8.1.12	Formatierung und Parsing	516
8.2	Datumsarithmetik	517
8.2.1	Einstieg in die Datumsarithmetik	517
8.2.2	Real-World-Example: Gehaltszahltag	522
8.3	Interoperabilität mit Legacy-Code	526

9	Applikationsbausteine	529
9.1	Einsatz von Bibliotheken	530
9.2	Google Guava im Kurzüberblick	533
9.2.1	String-Aktionen	534
9.2.2	Stringkonkatenation und -extraktion	535
9.2.3	Erweiterungen für Collections	540
9.2.4	Weitere Utility-Funktionalitäten	544
9.3	Wertebereichs- und Parameterprüfungen	548
9.4	Logging-Frameworks	551
9.4.1	Apache log4j2	552
9.4.2	Tipps und Tricks zum Einsatz von Logging mit log4j2	556
9.5	Konfigurationsparameter und -dateien	561
9.5.1	Einlesen von Kommandozeilenparametern	561
9.5.2	Verarbeitung von Properties	569
9.5.3	Weitere Möglichkeiten zur Konfigurationsverwaltung	575
10	Multithreading-Grundlagen	581
10.1	Threads und Runnables	583
10.1.1	Definition der auszuführenden Aufgabe	583
10.1.2	Start, Ausführung und Ende von Threads	584
10.1.3	Lebenszyklus von Threads und Thread-Zustände	588
10.1.4	Unterbrechungswünsche durch Aufruf von <code>interrupt()</code> ..	591
10.2	Zusammenarbeit von Threads	594
10.2.1	Konkurrierende Datenzugriffe	594
10.2.2	Locks, Monitore und kritische Bereiche	595
10.2.3	Deadlocks und Starvation	602
10.2.4	Kritische Bereiche und das Interface <code>Lock</code>	604
10.3	Kommunikation von Threads	606
10.3.1	Kommunikation mit Synchronisation	606
10.3.2	Kommunikation über die Methoden <code>wait()</code> , <code>notify()</code> und <code>notifyAll()</code>	610
10.3.3	Abstimmung von Threads	615
10.3.4	Unerwartete <code>IllegalMonitorStateException</code>	617
10.4	Das Java-Memory-Modell	619
10.4.1	Sichtbarkeit	619
10.4.2	Atomarität	620
10.4.3	Reorderings	622
10.5	Besonderheiten bei Threads	626
10.5.1	Verschiedene Arten von Threads	626
10.5.2	Exceptions in Threads	627
10.5.3	Sicheres Beenden von Threads	628
10.6	Weiterführende Literatur	632

11	Modern Concurrency	633
11.1	Concurrent Collections	634
11.1.1	Thread-Sicherheit und Parallelität mit »normalen« Collections	634
11.1.2	Parallelität mit den Concurrent Collections	636
11.1.3	Blockierende Warteschlangen und das Interface <code>BlockingQueue<E></code>	639
11.2	Das Executor-Framework	642
11.2.1	Einführung	642
11.2.2	Definition von Aufgaben	645
11.2.3	Parallele Abarbeitung im <code>ExecutorService</code>	649
11.3	Das Fork-Join-Framework	652
11.3.1	Einführendes Beispiel	653
11.3.2	Real-World-Example: Merge Sort	656
11.4	Die Klasse <code>CompletableFuture<T></code>	659
11.4.1	Einführung	659
11.4.2	Beispiel: Parallele Verarbeitung von Dateiinhalten	662
11.4.3	Erweiterungen in JDK 9	665
11.4.4	Beispiel: Von synchron zu multithreaded	667
11.5	Reactive Streams und die Klasse <code>Flow</code>	670
11.5.1	Schnelleinstieg Reactive Streams	670
11.5.2	Reactive Streams im JDK	671
11.5.3	Beispiel zur Klasse <code>Flow</code>	675
11.5.4	Fazit	679
11.6	Weiterführende Literatur	680
12	Fortgeschrittene Java-Themen	681
12.1	Crashkurs Reflection	681
12.1.1	Grundlagen	683
12.1.2	Zugriff auf Methoden und Attribute	686
12.1.3	Spezialfälle	691
12.1.4	Type Erasure und Typinformationen bei Generics	697
12.1.5	Fazit	699
12.2	Annotations	699
12.2.1	Einführung in Annotations	700
12.2.2	Standard-Annotations des JDKs	701
12.2.3	Definition eigener Annotations	703
12.2.4	Annotations zur Laufzeit auslesen	706
12.3	Serialisierung	708
12.3.1	Grundlagen der Serialisierung	708
12.3.2	Die Serialisierung anpassen	713
12.3.3	Versionsverwaltung der Serialisierung	716
12.3.4	Optimierung der Serialisierung	720

12.4	Garbage Collection	725
12.4.1	Grundlagen zur Garbage Collection	725
12.4.2	Der Garbage Collector »G1«	727
12.5	Dynamic Proxies	728
12.5.1	Statischer Proxy	730
12.5.2	Dynamischer Proxy	732
12.6	HTTP/2-API	737
12.6.1	Einführung	737
12.6.2	Real-World-Example: Wechselkurs mit REST	740
12.6.3	Fazit	742
12.7	Weiterführende Literatur	742
13	Basiswissen Internationalisierung	743
13.1	Internationalisierung im Überblick	743
13.1.1	Grundlagen und Normen	744
13.1.2	Die Klasse <code>Locale</code>	745
13.1.3	Die Klasse <code>PropertyResourceBundle</code>	749
13.1.4	Formatierte Ein- und Ausgabe	752
13.1.5	Datumswerte und die Klasse <code>DateFormat</code>	753
13.1.6	Zahlen und die Klasse <code>NumberFormat</code>	754
13.1.7	Textmeldungen und die Klasse <code>MessageFormat</code>	757
13.1.8	Stringvergleiche mit der Klasse <code>Collator</code>	759
13.2	Programmbausteine zur Internationalisierung	764
13.2.1	Unterstützung mehrerer Datumsformate	765
13.2.2	Fazit und Ausblick	770
III	Wichtige Neuerungen in Java 12 bis 15	771
14	Neues und Änderungen in den Java-Versionen 12 bis 15	773
14.1	Syntaxneuerungen	774
14.1.1	Text Blocks	774
14.1.2	Switch Expressions	779
14.1.3	Records (Preview)	786
14.1.4	Pattern Matching bei <code>instanceof</code> (Preview)	793
14.1.5	Sealed Types (Preview)	795
14.1.6	Lokale Enums und Interfaces (Preview)	797
14.2	API-Neuerungen	798
14.2.1	Neue Methoden in der Klasse <code>String</code>	798
14.2.2	Neue Hilfsmethode in der Utility-Klasse <code>Files</code>	801
14.2.3	Der <code>teeing()</code> -Kollektor	802

14.3	JVM-Neuerungen	805
14.3.1	Verbesserung bei <code>NullPointerException</code>	805
14.3.2	Entfernung der JavaScript-Engine	808
14.4	Microbenchmark Suite	808
14.4.1	Eigene Microbenchmarks und Varianten davon	808
14.4.2	Microbenchmarks mit JMH	811
14.4.3	Fazit zu JMH	817
14.5	Java 15 – notwendige Anpassungen für Build-Tools und IDEs	818
14.5.1	Java 15 mit Gradle	818
14.5.2	Java 15 mit Maven	819
14.5.3	Java 15 mit Eclipse	820
14.5.4	Java 15 mit IntelliJ	821
14.5.5	Java 15 mit JShell oder der Kommandozeile	821
14.6	Fazit	822

IV Modularisierung 823

15	Modularisierung mit Project Jigsaw	825
15.1	Grundlagen	826
15.1.1	Begrifflichkeiten	826
15.1.2	Ziele von Project Jigsaw	827
15.2	Modularisierung im Überblick	828
15.2.1	Grundlagen zu Project Jigsaw	828
15.2.2	Beispiel mit zwei Modulen	834
15.2.3	Packaging	843
15.2.4	Abhängigkeiten und Modulgraphen	845
15.2.5	Module des JDKs einbinden	847
15.2.6	Arten von Modulen	853
15.3	Sichtbarkeiten und Zugriffsschutz	854
15.3.1	Sichtbarkeiten	854
15.3.2	Zugriffsschutz und Reflection	856
15.4	Empfehlenswertes Verzeichnislayout für Module	858
15.5	Kompatibilität und Migration	860
15.5.1	Kompatibilitätsmodus	860
15.5.2	Migrationsszenarien	863
15.5.3	Fallstrick bei der Bottom-up-Migration	867
15.5.4	Beispiel: Migration mit Automatic Modules	868
15.5.5	Beispiel: Automatic und Unnamed Module	870
15.5.6	Beispiel: Abwandlung mit zwei Automatic Modules	873
15.5.7	Fazit	875
15.6	Zusammenfassung	876

V Fallstricke und Lösungen im Praxisalltag

877

16	Bad Smells	879
16.1	Programmdesign	881
16.1.1	Bad Smell: Verwenden von Magic Numbers	881
16.1.2	Bad Smell: Konstanten in Interfaces definieren	882
16.1.3	Bad Smell: Zusammengehörende Konstanten nicht als Typ definiert	884
16.1.4	Bad Smell: Casts auf unbekannte Subtypen	886
16.1.5	Bad Smell: Programmcode im Logging-Code	888
16.1.6	Bad Smell: Dominanter Logging-Code	889
16.1.7	Bad Smell: Unvollständige Änderungen nach Copy-Paste ..	890
16.1.8	Bad Smell: Unvollständige Betrachtung aller Alternativen ..	892
16.1.9	Bad Smell: Prä-/Postinkrement in komplexeren Statements	893
16.1.10	Bad Smell: Mehrere aufeinanderfolgende Parameter gleichen Typs	896
16.1.11	Bad Smell: Grundloser Einsatz von Reflection	897
16.1.12	Bad Smell: <code>System.exit()</code> mitten im Programm	898
16.1.13	Bad Smell: Variablendeklaration nicht im kleinstmöglichen Sichtbarkeitsbereich	900
16.2	Klassendesign	902
16.2.1	Bad Smell: Unnötigerweise veränderliche Attribute	902
16.2.2	Bad Smell: Herausgabe von <code>this</code> im Konstruktor	904
16.2.3	Bad Smell: Aufruf abstrakter Methoden im Konstruktor	905
16.2.4	Bad Smell: Mix abstrakter und konkreter Basisklassen	910
16.2.5	Bad Smell: Referenzierung von Subklassen in Basisklassen	912
16.2.6	Bad Smell: Öffentlicher Defaultkonstruktor lediglich zum Zugriff auf Hilfsmethoden	914
16.3	Fehlerbehandlung und Exception Handling	915
16.3.1	Bad Smell: Unbehandelte Exception	915
16.3.2	Bad Smell: Unpassender Exception-Typ	917
16.3.3	Bad Smell: Fangen der allgemeinsten Exception	918
16.3.4	Bad Smell: Exceptions zur Steuerung des Kontrollflusses ..	920
16.3.5	Bad Smell: Unbedachte Rückgabe von <code>null</code>	922
16.3.6	Bad Smell: Rückgabe von <code>null</code> statt Exception im Fehlerfall	924
16.3.7	Bad Smell: Sonderbehandlung von Randfällen	926
16.3.8	Bad Smell: Keine Gültigkeitsprüfung von Eingabeparametern	927
16.3.9	Bad Smell: Fehlerhafte Fehlerbehandlung	929
16.3.10	Bad Smell: I/O ohne <code>finally</code> oder ARM	931
16.3.11	Bad Smell: Resource Leaks durch Exceptions im Konstruktor	932
16.4	Häufige Fallstricke	937
16.5	Weiterführende Literatur	952

17	Refactorings	953
17.1	Refactorings am Beispiel	954
17.2	Das Standardvorgehen	962
17.3	Kombination von Basis-Refactorings	965
17.3.1	Refactoring-Beispiel: Ausgangslage und Ziel	965
17.3.2	Auflösen der Abhängigkeiten	967
17.3.3	Vereinfachungen	974
17.3.4	Verlagern von Funktionalität	978
17.4	Der Refactoring-Katalog	979
17.4.1	Reduziere die Sichtbarkeit von Attributen	979
17.4.2	Minimiere veränderliche Attribute	982
17.4.3	Reduziere die Sichtbarkeit von Methoden	986
17.4.4	Ersetze Mutator- durch Business-Methode	988
17.4.5	Minimiere Zustandsänderungen	989
17.4.6	Führe ein Interface ein	989
17.4.7	Spalte ein Interface auf	990
17.4.8	Führe ein Read-only-Interface ein	991
17.4.9	Führe ein Read-Write-Interface ein	992
17.4.10	Lagere Funktionalität in Hilfsmethoden aus	993
17.4.11	Trenne Informationsbeschaffung und -verarbeitung	995
17.4.12	Wandle Konstantensammlung in <code>enum</code> um	1002
17.4.13	Entferne Exceptions zur Steuerung des Kontrollflusses	1004
17.4.14	Wandle in Utility-Klasse mit statischen Hilfsmethoden um	1007
17.4.15	Löse <code>if-else / instanceof</code> durch Polymorphie auf	1010
17.5	Defensives Programmieren	1013
17.5.1	Führe eine Zustandsprüfung ein	1013
17.5.2	Überprüfe Eingabeparameter	1014
17.6	Fallstricke bei Refactorings	1019
17.7	Weiterführende Literatur	1021
18	Entwurfsmuster	1023
18.1	Erzeugungsmuster	1026
18.1.1	Erzeugungsmethode	1026
18.1.2	Fabrikmethode (Factory Method)	1029
18.1.3	Erbauer (Builder)	1032
18.1.4	Singleton	1035
18.1.5	Prototyp (Prototype)	1040
18.2	Strukturmuster	1044
18.2.1	Fassade (Façade)	1044
18.2.2	Adapter	1047
18.2.3	Dekorierer (Decorator)	1049
18.2.4	Kompositum (Composite)	1052

VI Qualitätssicherungsmaßnahmen 1099

19	Programmierstil und Coding Conventions	1101
19.1	Grundregeln eines guten Programmierstils	1101
19.1.1	Keep It Human-Readable	1102
19.1.2	Keep It Simple And Short (KISS)	1102
19.1.3	Keep It Natural	1102
19.1.4	Keep It Clean	1103
19.2	Die Psychologie beim Sourcecode-Layout	1103
19.2.1	Gesetz der Ähnlichkeit	1103
19.2.2	Gesetz der Nähe	1105
19.3	Coding Conventions	1106
19.3.1	Grundlegende Namens- und Formatierungsregeln	1107
19.3.2	Namensgebung	1110
19.3.3	Dokumentation	1113
19.3.4	Programmdesign	1115
19.3.5	Klassendesign	1120
19.3.6	Parameterlisten	1124
19.3.7	Logik und Kontrollfluss	1125
19.4	Sourcecode-Prüfung mit Tools	1128
19.4.1	Metriken	1129
19.4.2	Sourcecode-Prüfung im Build-Prozess	1133
20	Unit Tests	1143
20.1	Testen im Überblick	1143
20.1.1	Was versteht man unter Testen?	1143
20.1.2	Testarten im Überblick	1144
20.1.3	Zuständigkeiten beim Testen	1147
20.1.4	Testen und Qualität	1150
20.2	Wissenswertes zu Testfällen	1154
20.2.1	Testfälle mit JUnit definieren	1154

20.2.2	Problem der Kombinatorik beim Bestimmen von Testfällen .	1163
20.3	Besondere Assertions und Annotations	1171
20.4	Parametrisierte Tests mit JUnit 5	1182
20.4.1	Einstieg	1182
20.4.2	Verbesserung des Tests der Rabattberechnung.....	1189
20.4.3	Praxisbeispiel: Berechnung in Testfall vereinfachen	1191
20.4.4	Praxis-Trickkiste	1194
20.5	Fortgeschrittene Unit-Test-Techniken	1201
20.5.1	Stellvertreterobjekte / Test-Doubles	1204
20.5.2	Vorarbeiten für das Testen mit Stubs und Mocks	1208
20.5.3	Die Technik EXTRACT AND OVERRIDE	1210
20.5.4	Einstieg in das Testen mit Mocks und Mockito	1217
20.5.5	Abhängigkeiten mit Mockito auflösen	1226
20.5.6	Unit Tests von privaten Methoden.....	1228
20.6	Test Smells	1230
20.6.1	Test Smell: Unangebrachtes <code>assertTrue()</code> und <code>assertFalse()</code>	1230
20.6.2	Test Smell: Zu viele Asserts im Testfall	1231
20.6.3	Test Smell: Asserts ohne Hinweis.....	1233
20.6.4	Test Smell: Einsatz von <code>toString()</code> in <code>assertEquals()</code>	1234
20.6.5	Test Smell: Unit Tests zur Prüfung von Laufzeiten	1235
20.7	Nützliche Tools für Unit Tests	1237
20.7.1	Hamcrest	1237
20.7.2	AssertJ	1241
20.7.3	MoreUnit	1246
20.7.4	Infinittest	1247
20.7.5	JaCoCo	1248
20.7.6	EclEmma	1252
20.8	Umstieg von JUnit 4 auf JUnit 5	1254
20.8.1	Erweiterung im Gradle-Build für JUnit-5-Tests	1254
20.8.2	Veränderungen in den Annotations.....	1254
20.8.3	Alternativen für JUnit Rules	1256
20.8.4	Veränderungen bei parametrisierten Tests.....	1260
20.8.5	Alternative zur Hamcrest-Integration in JUnit 4	1261
20.9	Fazit	1262
20.10	Weiterführende Literatur	1263
21	Codereviews	1265
21.1	Definition	1265
21.2	Probleme und Tipps zur Durchführung.....	1267
21.3	Vorteile von Codereviews	1269
21.4	Codereview-Checkliste	1272

22	Optimierungen	1273
22.1	Grundlagen	1274
22.1.1	Optimierungsebenen und Einflussfaktoren	1275
22.1.2	Optimierungstechniken	1276
22.1.3	CPU-bound-Optimierungsebenen am Beispiel	1278
22.1.4	Messungen – Erkennen kritischer Bereiche	1282
22.1.5	Abschätzungen mit der O-Notation	1289
22.2	Einsatz geeigneter Datenstrukturen	1292
22.2.1	Einfluss von Arrays und Listen	1293
22.2.2	Optimierungen für <code>Set</code> und <code>Map</code>	1297
22.2.3	Design eines Zugriffsinterface	1299
22.3	Lazy Initialization	1302
22.3.1	Konsequenzen des Einsatzes der Lazy Initialization	1305
22.3.2	Lazy Initialization mithilfe des PROXY-Musters	1307
22.4	Optimierungen am Beispiel	1310
22.5	I/O-bound-Optimierungen	1318
22.5.1	Technik – Wahl passender Strategien	1318
22.5.2	Technik – Caching und Pooling	1321
22.5.3	Technik – Vermeidung unnötiger Aktionen	1321
22.6	Memory-bound-Optimierungen	1325
22.6.1	Technik – Wahl passender Strategien	1325
22.6.2	Technik – Caching und Pooling	1326
22.6.3	Optimierungen der Stringverarbeitung	1333
22.6.4	Technik – Vermeidung unnötiger Aktionen	1336
22.7	CPU-bound-Optimierungen	1338
22.7.1	Technik – Wahl passender Strategien	1339
22.7.2	Technik – Caching und Pooling	1342
22.7.3	Technik – Vermeidung unnötiger Aktionen	1344
22.8	Weiterführende Literatur	1347
23	Schlussgedanken	1349
 VII Anhang		 1351
A	Grundlagen zur Java Virtual Machine	1353
A.1	Wissenswertes rund um die Java Virtual Machine	1353
A.1.1	Ausführung eines Java-Programms	1353
A.1.2	Speicherverwaltung und Classloading	1354
Literaturverzeichnis		1357
Index		1361