

Inhaltsübersicht

1	Einleitung	1
Java-Grundlagen, Analyse und Design		9
2	Professionelle Arbeitsumgebung	9
3	Objektorientiertes Design	83
4	Java-Grundlagen	197
Bausteine stabiler Java-Applikationen		313
5	Das Collections-Framework	313
6	Applikationsbausteine	423
7	Multithreading	485
8	Fortgeschrittene Java-Themen	573
9	Programmierung grafischer Benutzeroberflächen mit Swing ..	639
10	Basiswissen Internationalisierung	755
Die Neuerungen in Java 8		795
11	Lambda-Ausdrücke	795
12	Bulk Operations on Collections	815
13	JSR-310: Date And Time API	859
14	Einstieg JavaFX 8	875
15	Weitere Änderungen in JDK 8	917

Fallstricke und Lösungen im Praxisalltag		951
16	Bad Smells	951
17	Refactorings	1021
18	Entwurfsmuster	1085
Qualitätssicherungsmaßnahmen		1155
19	Programmierstil und Coding Conventions	1155
20	Unit Tests	1195
21	Codereviews	1279
22	Optimierungen	1289
Anhang		1363
Literaturverzeichnis		1365
Index		1369

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

Java-Grundlagen, Analyse und Design	9
--------------------------------------------	----------

2	Professionelle Arbeitsumgebung	9
2.1	Vorteile von IDEs am Beispiel von Eclipse	10
2.2	Projektorganisation	12
2.2.1	Projektstruktur in Eclipse und Ant	12
2.2.2	Projektstruktur für Maven und Gradle	14
2.3	Einsatz von Versionsverwaltungen	16
2.3.1	Arbeiten mit zentralen Versionsverwaltungen	19
2.3.2	Dezentrale Versionsverwaltungen	24
2.3.3	VCS und DVCS im Vergleich	30
2.4	Einsatz eines Unit-Test-Frameworks	33
2.4.1	Das JUnit-Framework	33
2.4.2	Vorteile von Unit Tests	38
2.5	Debugging	39
2.5.1	Fehlersuche mit einem Debugger	41
2.5.2	Remote Debugging	44
2.6	Deployment von Java-Applikationen	48
2.6.1	Das JAR-Tool im Kurzüberblick	50
2.6.2	JAR inspizieren und ändern, Inhalt extrahieren	51
2.6.3	Metainformationen und das Manifest	52
2.6.4	Inspizieren einer JAR-Datei	55

2.7	Einsatz eines IDE-unabhängigen Build-Prozesses	57
2.7.1	Ant im Überblick	60
2.7.2	Maven im Überblick	62
2.7.3	Builds mit Gradle	68
2.7.4	Vergleich von Ant, Maven und Gradle	80
2.8	Weiterführende Literatur	81
3	Objektorientiertes Design	83
3.1	OO-Grundlagen	84
3.1.1	Grundbegriffe	84
3.1.2	Beispielentwurf: Ein Zähler	96
3.1.3	Vom imperativen zum objektorientierten Entwurf	104
3.1.4	Diskussion der OO-Grundgedanken	109
3.1.5	Wissenswertes zum Objektzustand	112
3.2	Grundlegende OO-Techniken	121
3.2.1	Schnittstellen (Interfaces)	121
3.2.2	Basisklassen und abstrakte Basisklassen	126
3.2.3	Interfaces und abstrakte Basisklassen	128
3.3	Wissenswertes zu Vererbung	130
3.3.1	Probleme durch Vererbung	130
3.3.2	Delegation statt Vererbung	136
3.4	Fortgeschrittenere OO-Techniken	140
3.4.1	Read-only-Interface	140
3.4.2	Immutable-Klasse	146
3.4.3	Marker-Interface	151
3.4.4	Konstantensammlungen und Aufzählungen	152
3.4.5	Value Object (Data Transfer Object)	157
3.5	Prinzipien guten OO-Designs	159
3.5.1	Geheimnisprinzip nach Parnas	160
3.5.2	Law of Demeter	161
3.5.3	SOLID-Prinzipien	164
3.6	Formen der Varianz	177
3.6.1	Grundlagen der Varianz	177
3.6.2	Kovariante Rückgabewerte	180
3.7	Generische Typen (Generics)	182
3.7.1	Einführung	183
3.7.2	Generics und Auswirkungen der Type Erasure	188
3.8	Weiterführende Literatur	196
4	Java-Grundlagen	197
4.1	Die Klasse Object	197
4.1.1	Die Methode toString()	199
4.1.2	Die Methode equals()	203

4.2	Primitive Typen und Wrapper-Klassen	215
4.2.1	Grundlagen	216
4.2.2	Konvertierung von Werten	223
4.2.3	Wissenswertes zu Auto-Boxing und Auto-Unboxing	226
4.2.4	Ausgabe und Verarbeitung von Zahlen	229
4.3	Stringverarbeitung	232
4.3.1	Die Klasse <code>String</code>	233
4.3.2	Die Klassen <code>StringBuffer</code> und <code>StringBuilder</code>	237
4.3.3	Ausgaben mit <code>format()</code> und <code>printf()</code>	241
4.3.4	Die Klasse <code>StringTokenizer</code>	242
4.3.5	Die Methode <code>split()</code> und das 1x1 der regulären Ausdrücke	244
4.4	Datumsverarbeitung	250
4.4.1	Fallstricke der Datums-APIs	250
4.4.2	Das <code>Date</code> -API	252
4.4.3	Das <code>Calendar</code> -API	254
4.5	Innere Interfaces und innere Klassen	256
4.5.1	Varianten innerer Klassen	257
4.5.2	Innere Interfaces	260
4.5.3	Designbeispiel mit inneren Klassen und Interfaces	261
4.5.4	Lokal definierte Klassen und Interfaces	263
4.6	Ein- und Ausgabe (I/O)	266
4.6.1	Dateibehandlung und die Klasse <code>File</code>	266
4.6.2	Ein- und Ausgabestreams im Überblick	273
4.6.3	Zeichencodierungen bei der Ein- und Ausgabe	276
4.6.4	Speichern und Laden von Daten und Objekten	278
4.6.5	Dateiverarbeitung in JDK 7	287
4.7	Fehlerbehandlung	292
4.7.1	Einstieg in die Fehlerbehandlung	292
4.7.2	Checked Exceptions und Unchecked Exceptions	297
4.7.3	Exception Handling und Ressourcenfreigabe	298
4.7.4	Besonderheiten beim Exception Handling mit JDK 7	304
4.7.5	Assertions	308
4.8	Weiterführende Literatur	312

Bausteine stabiler Java-Applikationen

313

5	Das Collections-Framework	313
5.1	Datenstrukturen und Containerklasse	313
5.1.1	Wahl einer geeigneten Datenstruktur	314
5.1.2	Arrays	316
5.1.3	Das Interface <code>Collection</code>	319

5.1.4	Das Interface <i>Iterator</i>	320
5.1.5	Listen und das Interface <i>List</i>	323
5.1.6	Mengen und das Interface <i>Set</i>	330
5.1.7	Grundlagen von hashbasierten Containern	336
5.1.8	Grundlagen automatisch sortierender Container	346
5.1.9	Die Methoden <i>equals()</i> , <i>hashCode()</i> und <i>compareTo()</i> im Zusammenspiel	354
5.1.10	Schlüssel-Wert-Abbildungen und das Interface <i>Map</i>	356
5.1.11	Erweiterungen am Beispiel der Klasse <i>HashMap</i>	364
5.1.12	Entscheidungshilfe zur Wahl von Datenstrukturen	369
5.2	Suchen, Sortieren und Filtern	370
5.2.1	Suchen	370
5.2.2	Sortieren von Arrays und Listen	373
5.2.3	Sortieren mit Komparatoren	375
5.2.4	Filtern von Collections	381
5.3	Utility-Klassen und Hilfsmethoden	387
5.3.1	Nützliche Hilfsmethoden	388
5.3.2	Dekorierer <i>synchronized</i> , <i>unmodifiable</i> und <i>checked</i>	391
5.3.3	Vordefinierte Algorithmen	397
5.4	Containerklassen: Generics und Varianz	400
5.5	Fallstricke im Collections-Framework	414
5.5.1	Wissenswertes zu Arrays	414
5.5.2	Wissenswertes zu <i>Stack</i> , <i>Queue</i> und <i>Deque</i>	417
5.6	Weiterführende Literatur	421
6	Applikationsbausteine	423
6.1	Einsatz von Bibliotheken	424
6.2	Google Guava im Kurzüberblick	427
6.2.1	String-Aktionen	429
6.2.2	String-Konkatenation und -Extraktion	430
6.2.3	Erweiterungen für Collections	434
6.2.4	Weitere Utility-Funktionalitäten	438
6.3	Wertebereichs- und Parameterprüfungen	444
6.3.1	Prüfung einfacher Wertebereiche und Wertemengen	445
6.3.2	Prüfung komplexerer Wertebereiche	447
6.4	Logging-Frameworks	450
6.4.1	Apache log4j	451
6.4.2	Tipps und Tricks zum Einsatz von Logging mit log4j	458
6.5	Konfigurationsparameter und -dateien	463
6.5.1	Einlesen von Kommandozeilenparametern	463
6.5.2	Verarbeitung von Properties	471
6.5.3	Die Klasse <i>Preferences</i>	478
6.5.4	Weitere Möglichkeiten zur Konfigurationsverwaltung	479

7	Multithreading	485
7.1	Threads und Runnables	487
7.1.1	Definition der auszuführenden Aufgabe	487
7.1.2	Start, Ausführung und Ende von Threads	488
7.1.3	Lebenszyklus von Threads und Thread-Zustände	492
7.1.4	Unterbrechungswünsche durch Aufruf von <code>interrupt()</code>	495
7.2	Zusammenarbeit von Threads	497
7.2.1	Konkurrierende Datenzugriffe	497
7.2.2	Locks, Monitore und kritische Bereiche	499
7.2.3	Deadlocks und Starvation	506
7.2.4	Kritische Bereiche und das Interface <code>Lock</code>	508
7.3	Kommunikation von Threads	513
7.3.1	Kommunikation mit Synchronisation	514
7.3.2	Kommunikation über die Methoden <code>wait()</code> , <code>notify()</code> und <code>notifyAll()</code>	517
7.3.3	Abstimmung von Threads	526
7.3.4	Unerwartete <code>IllegalMonitorStateException</code> s	530
7.4	Das Java-Memory-Modell	532
7.4.1	Sichtbarkeit	533
7.4.2	Atomarität	533
7.4.3	Reorderings	535
7.5	Besonderheiten bei Threads	538
7.5.1	Verschiedene Arten von Threads	539
7.5.2	Exceptions in Threads	539
7.5.3	Sicheres Beenden von Threads	541
7.5.4	Zeitgesteuerte Ausführung	544
7.6	Die Concurrency Utilities	548
7.6.1	Concurrent Collections	549
7.6.2	Das Executor-Framework	557
7.6.3	Das Fork-Join-Framework	568
7.7	Weiterführende Literatur	570
8	Fortgeschrittene Java-Themen	573
8.1	Crashkurs Reflection	573
8.1.1	Grundlagen	575
8.1.2	Zugriff auf Methoden und Attribute	578
8.1.3	Spezialfälle	583
8.1.4	Type Erasure und Typinformationen bei Generics	586
8.2	Annotations	588
8.2.1	Einführung in Annotations	589
8.2.2	Standard-Annotations des JDKs	590
8.2.3	Definition eigener Annotations	592
8.2.4	Annotation zur Laufzeit auslesen	595

8.3	Serialisierung	596
8.3.1	Grundlagen der Serialisierung	597
8.3.2	Die Serialisierung anpassen	602
8.3.3	Versionsverwaltung der Serialisierung	605
8.3.4	Optimierung der Serialisierung	609
8.4	Objektkopien und das Interface Cloneable	614
8.4.1	Das Interface Cloneable	614
8.4.2	Alternativen zur Methode clone()	623
8.5	Garbage Collection	625
8.5.1	Grundlagen zur Garbage Collection	626
8.5.2	Herkömmliche Algorithmen zur Garbage Collection	629
8.5.3	Einflussfaktoren auf die Garbage Collection	631
8.5.4	Der Garbage Collector »G1«	633
8.5.5	Memory Leaks: Gibt es die auch in Java?!	634
8.5.6	Objektzerstörung und finalize()	636
8.6	Weiterführende Literatur	638
9	Programmierung grafischer Benutzeroberflächen mit Swing ..	639
9.1	Grundlagen zu grafischen Oberflächen	640
9.1.1	Überblick: Bedienelemente und Container	644
9.1.2	Einführung in das Layoutmanagement	646
9.1.3	Komplexere Layouts (Kombination von Layoutmanagern) ..	651
9.1.4	Grundlagen zur Ereignisbehandlung	655
9.1.5	Weitere gebräuchliche Event Listener	660
9.1.6	Varianten der Ereignisverarbeitung	665
9.2	Multithreading und Swing	670
9.2.1	Crashkurs Event Handling in Swing	670
9.2.2	Ausführen von Aktionen	672
9.2.3	Die Klasse SwingWorker	675
9.3	Zeichnen in GUI-Komponenten	679
9.3.1	Generelles zum Zeichnen in GUI-Komponenten	679
9.3.2	Kurzeinführung in Java 2D	687
9.3.3	Bedienelemente mit Java 2D selbst erstellen	692
9.4	Komplexe Bedienelemente	699
9.4.1	Grundlagen	699
9.4.2	Die Klasse JList	709
9.4.3	Die Klasse JTable	723
9.4.4	Die Klasse JTree	743
9.5	Weiterführende Literatur	753

10	Basiswissen Internationalisierung	755
10.1	Internationalisierung im Überblick	755
10.1.1	Grundlagen und Normen	756
10.1.2	Die Klasse <code>Locale</code>	757
10.1.3	Die Klasse <code>PropertyResourceBundle</code>	760
10.1.4	Formatierte Ein- und Ausgabe	764
10.1.5	Zahlen und die Klasse <code>NumberFormat</code>	765
10.1.6	Datumswerte und die Klasse <code>DateFormat</code>	768
10.1.7	Textmeldungen und die Klasse <code>MessageFormat</code>	773
10.1.8	Stringvergleiche mit der Klasse <code>Collator</code>	775
10.2	Programmbausteine zur Internationalisierung	780
10.2.1	Unterstützung mehrerer Datumsformate	781
10.2.2	Nutzung mehrerer Sprachdateien	785

Die Neuerungen in Java 8	795
---------------------------------	------------

11	Lambda-Ausdrücke	795
11.1	Einstieg in Lambdas	796
11.1.1	Lambdas am Beispiel	796
11.1.2	Functional Interfaces und SAM-Typen	797
11.1.3	Type Inference und Kurzformen der Syntax	800
11.1.4	Lambdas als Parameter und als Rückgabewerte	801
11.1.5	Unterschiede: Lambdas vs. anonyme innere Klassen	801
11.2	Defaultmethoden	803
11.2.1	Interface-Erweiterungen	804
11.2.2	Vorgabe von Standardverhalten	806
11.2.3	Erweiterte Möglichkeiten durch Defaultmethoden	807
11.2.4	Spezialfall: Was passiert bei Konflikten?	808
11.2.5	Vorteile und Gefahren von Defaultmethoden	809
11.2.6	Statische Methoden in Interfaces	810
11.3	Methodenreferenzen	812
11.4	Fazit	814
12	Bulk Operations on Collections	815
12.1	Externe vs. interne Iteration	815
12.1.1	Externe Iteration	816
12.1.2	Interne Iteration	816
12.1.3	Externe vs. interne Iteration an einem Beispiel	817
12.2	Collections-Erweiterungen	819
12.2.1	Das Interface <code>Predicate<T></code>	819
12.2.2	Die Methode <code>Collection.removeIf()</code>	821
12.2.3	Das Interface <code>UnaryOperator<T></code>	822
12.2.4	Die Methode <code>List.replaceAll()</code>	824

12.3	Streams	824
12.3.1	Streams erzeugen — Create Operations	825
12.3.2	Intermediate und Terminal Operations im Überblick	829
12.3.3	Zustandslose Intermediate Operations	831
12.3.4	Zustandsbehaftete Intermediate Operations	839
12.3.5	Terminal Operations	840
12.3.6	Wissenswertes zur Parallelverarbeitung	848
12.4	Filter-Map-Reduce	849
12.4.1	Herkömmliche Realisierung	849
12.4.2	Filter-Map-Reduce mit JDK 8	851
12.5	Fallstricke bei Lambdas und funktionaler Programmierung	854
12.5.1	Java-Fehlermeldungen werden zu komplex	854
12.5.2	Fallstrick: Imperative Lösung 1:1 funktional umsetzen	856
12.6	Fazit.....	857
13	JSR-310: Date And Time API	859
13.1	Datumsverarbeitung vor JSR-310	859
13.2	Überblick über die neu eingeführten Klassen	862
13.2.1	Die Klasse Instant	862
13.2.2	Die Aufzählung ChronoUnit	863
13.2.3	Die Klasse Duration	864
13.2.4	Die Klassen LocalDate, LocalTime und LocalDate- Time	866
13.2.5	Die Aufzählungen DayOfWeek und Month	867
13.2.6	Die Klassen YearMonth, MonthDay und Year	868
13.2.7	Die Klasse Period.....	869
13.2.8	Die Klasse Clock.....	871
13.2.9	Die Klasse ZonedDateTime	871
13.2.10	Beispiel: Berechnung einer Zeitdifferenz	872
13.2.11	Interoperabilität mit Legacy-Code	873
13.3	Fazit.....	874
14	Einstieg JavaFX 8	875
14.1	Einführung – JavaFX im Überblick	875
14.1.1	Motivation für JavaFX und Historisches	875
14.1.2	Grundsätzliche Konzepte	876
14.1.3	Layoutmanagement	880
14.2	Deklarativer Aufbau des GUIs	890
14.2.1	Deklarative Beschreibung von GUIs	890
14.2.2	Hello-World-Beispiel mit FXML	890
14.2.3	Diskussion: Design und Funktionalität strikt trennen	893

14.3	Rich-Client Experience	895
14.3.1	Gestaltung mit CSS	895
14.3.2	Effekte	901
14.3.3	Animationen	903
14.4	Neuerungen in JavaFX 8	905
14.4.1	Unterstützung von Lambdas als <code>EventHandler</code>	905
14.4.2	Texteffekte	906
14.4.3	Neue Controls	907
14.4.4	JavaFX 3D	914
14.5	Fazit	916
15	Weitere Änderungen in JDK 8	917
15.1	Erweiterungen im Interface <code>Comparator<T></code>	917
15.2	Die Klasse <code>Optional<T></code>	923
15.2.1	Grundlagen zur Klasse <code>Optional<T></code>	923
15.2.2	Weiterführendes Beispiel und Diskussion	926
15.3	Parallele Operationen auf Arrays	928
15.4	Erweiterungen im Interface <code>Map<K, V></code>	932
15.5	Erweiterungen im NIO und der Klasse <code>Files</code>	936
15.6	Erweiterungen im Bereich Concurrency	938
15.7	»Nashorn« – die neue JavaScript-Engine	942
15.8	Keine Permanent Generation mehr	945
15.9	Erweiterungen im Bereich Reflection	946
15.10	Base64-Codierungen	948
15.11	Änderungen bei Annotations	949

Fallstricke und Lösungen im Praxisalltag	951
-------------------------------------------------	------------

16	Bad Smells	951
16.1	Programmdesign	953
16.1.1	Bad Smell: Verwenden von Magic Numbers	953
16.1.2	Bad Smell: Konstanten in Interfaces definieren	954
16.1.3	Bad Smell: Zusammengehörende Konstanten nicht als Typ definiert	956
16.1.4	Bad Smell: Programmcode im Logging-Code	958
16.1.5	Bad Smell: Dominanter Logging-Code	959
16.1.6	Bad Smell: Unvollständige Betrachtung aller Alternativen ..	961
16.1.7	Bad Smell: Unvollständige Änderungen nach Copy-Paste ..	962
16.1.8	Bad Smell: Casts auf unbekannte Subtypen	964
16.1.9	Bad Smell: Pre-/Post-Increment in komplexeren Statements ..	965
16.1.10	Bad Smell: Keine Klammern um Blöcke	967

16.1.11	Bad Smell: Mehrere aufeinanderfolgende Parameter gleichen Typs	969
16.1.12	Bad Smell: Grundloser Einsatz von Reflection	970
16.1.13	Bad Smell: <code>System.exit()</code> mitten im Programm	972
16.1.14	Bad Smell: Variablendeklaration nicht im kleinstmöglichen Sichtbarkeitsbereich	973
16.2	Klassendesign	975
16.2.1	Bad Smell: Unnötigerweise veränderliche Attribute	975
16.2.2	Bad Smell: Herausgabe von <code>this</code> im Konstruktor	977
16.2.3	Bad Smell: Aufruf abstrakter Methoden im Konstruktor ...	979
16.2.4	Bad Smell: Referenzierung von Subklassen in Basisklassen	983
16.2.5	Bad Smell: Mix abstrakter und konkreter Basisklassen ...	985
16.2.6	Bad Smell: Öffentlicher Defaultkonstruktor lediglich zum Zugriff auf Hilfsmethoden	987
16.3	Fehlerbehandlung und Exception Handling	989
16.3.1	Bad Smell: Unbehandelte Exception	989
16.3.2	Bad Smell: Unpassender Exception-Typ	990
16.3.3	Bad Smell: Exceptions zur Steuerung des Kontrollflusses	992
16.3.4	Bad Smell: Fangen der allgemeinsten Exception	993
16.3.5	Bad Smell: Rückgabe von <code>null</code> statt Exception im Fehlerfall	995
16.3.6	Bad Smell: Unbedachte Rückgabe von <code>null</code>	996
16.3.7	Bad Smell: Sonderbehandlung von Randfällen	999
16.3.8	Bad Smell: Keine Gültigkeitsprüfung von Eingabeparametern	1000
16.3.9	Bad Smell: Fehlerhafte Fehlerbehandlung	1002
16.3.10	Bad Smell: I/O ohne <code>finally</code>	1004
16.3.11	Bad Smell: Resource Leaks durch Exceptions im Konstruktor	1005
16.4	Häufige Fallstricke	1010
16.5	Weiterführende Literatur	1020
17	Refactorings	1021
17.1	Refactorings am Beispiel	1022
17.2	Das Standardvorgehen	1030
17.3	Kombination von Basis-Refactorings	1033
17.3.1	Refactoring-Beispiel: Ausgangslage und Ziel	1033
17.3.2	Auflösen der Abhängigkeiten	1035
17.3.3	Vereinfachungen	1042
17.3.4	Verlagern von Funktionalität	1046
17.4	Der Refactoring-Katalog	1047
17.4.1	Reduziere die Sichtbarkeit von Attributen	1047
17.4.2	Minimiere veränderliche Attribute	1050

17.4.3	Reduziere die Sichtbarkeit von Methoden	1054
17.4.4	Ersetze Mutator- durch Business-Methode	1056
17.4.5	Minimiere Zustandsänderungen	1057
17.4.6	Führe ein Interface ein	1057
17.4.7	Spalte ein Interface auf	1058
17.4.8	Führe ein Read-only-Interface ein	1059
17.4.9	Führe ein Read-Write-Interface ein	1059
17.4.10	Lagere Funktionalität in Hilfsmethoden aus	1060
17.4.11	Trenne Informationsbeschaffung und -verarbeitung	1062
17.4.12	Wandle Konstantensammlung in enum um	1069
17.4.13	Entferne Exceptions zur Steuerung des Kontrollflusses ...	1072
17.4.14	Wandle in Utility-Klasse mit statischen Hilfsmethoden um	1074
17.5	Defensives Programmieren	1077
17.5.1	Führe eine Zustandsprüfung ein	1078
17.5.2	Überprüfe Eingabeparameter	1079
17.6	Weiterführende Literatur	1083
18	Entwurfsmuster	1085
18.1	Erzeugungsmuster	1088
18.1.1	Erzeugungsmethode	1088
18.1.2	Fabrikmethode (Factory method)	1091
18.1.3	Erbauer (Builder)	1094
18.1.4	Singleton	1097
18.1.5	Prototyp (Prototype)	1102
18.2	Strukturmuster	1106
18.2.1	Fassade (Façade)	1106
18.2.2	Adapter	1109
18.2.3	Dekorierer (Decorator)	1111
18.2.4	Kompositum (Composite)	1114
18.3	Verhaltensmuster	1118
18.3.1	Iterator	1118
18.3.2	Null-Objekt (Null Object)	1120
18.3.3	Schablonenmethode (Template method)	1123
18.3.4	Strategie (Strategy)	1127
18.3.5	Befehl (Command)	1135
18.3.6	Proxy	1142
18.3.7	Beobachter (Observer)	1144
18.3.8	MVC-Architektur	1152
18.4	Weiterführende Literatur	1154

Qualitätssicherungsmaßnahmen		1155
19	Programmierstil und Coding Conventions	1155
19.1	Grundregeln eines guten Programmierstils	1155
19.1.1	Keep It Human-Readable	1156
19.1.2	Keep It Simple And Short (KISS)	1156
19.1.3	Keep It Natural	1156
19.1.4	Keep It Clean	1157
19.2	Die Psychologie beim Sourcecode-Layout	1157
19.2.1	Gesetz der Ähnlichkeit	1157
19.2.2	Gesetz der Nähe	1159
19.3	Coding Conventions	1160
19.3.1	Grundlegende Namens- und Formatierungsregeln	1161
19.3.2	Namensgebung	1164
19.3.3	Dokumentation	1167
19.3.4	Programmdesign	1169
19.3.5	Klassendesign	1174
19.3.6	Parameterlisten	1177
19.3.7	Logik und Kontrollfluss	1179
19.4	Sourcecode-Prüfung mit Tools	1181
19.4.1	Metriken	1182
19.4.2	Sourcecode-Prüfung im Build-Prozess	1186
20	Unit Tests	1195
20.1	Testen im Überblick	1195
20.1.1	Was versteht man unter Testen?	1196
20.1.2	Testarten im Überblick	1197
20.1.3	Zuständigkeiten beim Testen	1199
20.1.4	Testen und Qualität	1201
20.2	Wissenswertes zu Testfällen	1205
20.2.1	Test-Driven Development (TDD) im Überblick	1205
20.2.2	Testfälle mit JUnit 4 definieren	1206
20.2.3	Problem der Komplexität	1213
20.3	Motivation für Unit Tests aus der Praxis	1217
20.3.1	Unit Tests für Weiterentwicklungen	1217
20.3.2	Unit Tests und Legacy-Code	1226
20.4	Fortgeschrittene Unit-Test-Techniken	1236
20.4.1	Test-Doubles	1236
20.4.2	Testen mit Stubs	1238
20.4.3	Testen mit Mocks	1240
20.4.4	Unit Tests von privaten Methoden	1242
20.5	Unit Tests mit Threads und Timing	1244
20.6	Test Smells	1249

20.7	JUnit Rules und parametrisierte Tests	1254
20.7.1	JUnit Rules im Überblick	1254
20.7.2	Parametrisierte Tests	1260
20.8	Nützliche Tools für Unit Tests	1263
20.8.1	Hamcrest	1263
20.8.2	MoreUnit	1269
20.8.3	Infinittest	1269
20.8.4	Cobertura	1270
20.8.5	EclEmma	1274
20.9	Schlussgedanken	1274
20.10	Weiterführende Literatur	1277
21	Codereviews	1279
21.1	Definition	1279
21.2	Probleme und Tipps zur Durchführung	1281
21.3	Vorteile von Codereviews	1283
21.4	Codereview-Tools	1286
21.5	Codereview-Checkliste	1288
22	Optimierungen	1289
22.1	Grundlagen	1290
22.1.1	Optimierungsebenen und Einflussfaktoren	1291
22.1.2	Optimierungstechniken	1292
22.1.3	CPU-bound-Optimierungsebenen am Beispiel	1294
22.1.4	Messungen – Erkennen kritischer Bereiche	1298
22.1.5	Abschätzungen mit der O-Notation	1305
22.2	Einsatz geeigneter Datenstrukturen	1308
22.2.1	Einfluss von Arrays und Listen	1309
22.2.2	Optimierungen für Set und Map	1313
22.2.3	Design eines Zugriffsinterface	1315
22.3	Lazy Initialization	1319
22.3.1	Lazy Initialization am Beispiel	1319
22.3.2	Konsequenzen des Einsatzes der Lazy Initialization	1322
22.3.3	Lazy Initialization mithilfe des PROXY-Musters	1324
22.4	Optimierungen am Beispiel	1327
22.5	I/O-bound-Optimierungen	1334
22.5.1	Technik – Wahl passender Strategien	1334
22.5.2	Technik – Caching und Pooling	1338
22.5.3	Technik – Vermeidung unnötiger Aktionen	1338
22.6	Memory-bound-Optimierungen	1341
22.6.1	Technik – Wahl passender Strategien	1341
22.6.2	Technik – Caching und Pooling	1344

22.6.3	Optimierungen der Stringverarbeitung	1350
22.6.4	Technik – Vermeidung unnötiger Aktionen	1352
22.7	CPU-bound-Optimierungen	1355
22.7.1	Technik – Wahl passender Strategien	1355
22.7.2	Technik – Caching und Pooling	1357
22.7.3	Technik – Vermeidung unnötiger Aktionen	1358
22.8	Weiterführende Literatur	1361

Anhang	1363
---------------	-------------

Literaturverzeichnis	1365
-----------------------------------	-------------

Index	1369
--------------------	-------------