

Inhaltsverzeichnis

1	Einleitung	xix
1.1	Über dieses Buch	xix
1.1.1	Motivation	xix
1.1.2	Was leistet dieses Buch und was nicht?	xx
1.1.3	Wie und was soll mithilfe des Buchs gelernt werden?	xx
1.1.4	Wer sollte dieses Buch lesen?	xxii
1.2	Aufbau des Buchs	xxii
1.2.1	Gliederung des Buchs	xxiii
1.2.2	Kapitelübersicht	xxiv
1.3	Konventionen	xxix
1.4	Danksagungen	xxx

I Java-Grundlagen, Analyse und Design **1**

2	Professionelle Arbeitsumgebung	3
2.1	Vorteile von IDEs am Beispiel von Eclipse	3
2.2	Projektorganisation	4
2.3	Einsatz von Versionsverwaltungen	6
2.3.1	Arbeiten mit Versionsverwaltungen	9
2.3.2	Tagging und Branching	10
2.3.3	CVS und SVN im Vergleich	13
2.4	Einsatz eines Unit-Test-Frameworks	14
2.4.1	Das JUnit-Framework	15
2.4.2	Schreiben und Ausführen von Tests	17
2.5	Einsatz eines IDE-unabhängigen Build-Prozesses	20
2.5.1	Motivation für ein IDE-unabhängiges Build-System	20
2.5.2	Ant-Build	21
2.6	Weiterführende Literatur	30
3	Objektorientiertes Design	31
3.1	OO-Grundlagen	32
3.1.1	Grundbegriffe	32

3.1.2	Beispielentwurf: Ein Zähler	40
3.1.3	Diskussion der OO-Grundgedanken	47
3.1.4	Wissenswertes zum Objektzustand	51
3.2	Grundlegende OO-Techniken	60
3.2.1	Abstrakte Basisklassen	60
3.2.2	Schnittstellen (Interfaces)	62
3.2.3	Interfaces und abstrakte Basisklassen	64
3.3	Vom imperativen zum objektorientierten Entwurf	65
3.4	Fortgeschrittenere OO-Techniken	69
3.4.1	Read-only-Interface	70
3.4.2	Immutable-Klasse	76
3.4.3	Delegation statt Vererbung	81
3.4.4	Marker-Interface	85
3.4.5	Konstantensammlungen, Aufzählungen und das Enum-Muster	86
3.4.6	Value Object	91
3.5	Formen der Varianz	94
3.5.1	Grundlagen der Varianz	94
3.5.2	Kovariante Rückgabewerte	97
3.6	Generische Typen (Generics)	99
3.6.1	Einführung	99
3.6.2	Generics und Auswirkungen der Type Erasure	104
3.6.3	Containerklassen: Generics und Varianz	111
3.7	Weiterführende Literatur	123
4	Java-Grundlagen	125
4.1	Die Klasse <code>Object</code>	125
4.1.1	Die Methode <code>toString()</code>	127
4.1.2	Die Methode <code>equals()</code>	131
4.2	Primitive Datentypen und Wrapper-Klassen	142
4.2.1	Konvertierung von Werten	143
4.2.2	Ausgabe und Verarbeitung von Zahlen	149
4.3	Stringverarbeitung	152
4.3.1	Die Klasse <code>String</code>	152
4.3.2	Die Klassen <code>StringBuffer</code> und <code>StringBuilder</code>	156
4.3.3	Ausgaben mit <code>format()</code> und <code>printf()</code>	159
4.3.4	Die Klasse <code>StringTokenizer</code>	161
4.3.5	Die Methode <code>split()</code> und das 1x1 der regulären Ausdrücke	162
4.4	Datumsverarbeitung	167
4.4.1	Fallstricke der Datums-APIs	168
4.4.2	Das <code>Date</code> -API	169
4.4.3	Das <code>Calendar</code> -API	172

4.5	Interfaces und innere Klassen	174
4.5.1	Interfaces	174
4.5.2	Varianten innerer Klassen	175
4.5.3	Designbeispiel mit inneren Klassen und Interfaces	178
4.5.4	Lokal definierte Klassen und Interfaces	180
4.6	Ein- und Ausgabe (I/O)	183
4.6.1	Dateibehandlung und die Klasse <code>File</code>	183
4.6.2	Ein- und Ausgabestreams	190
4.6.3	Speichern und Laden von Daten und Objekten	197
4.6.4	Grundlagen der Netzwerkprogrammierung	204
4.7	Fehlerbehandlung	209
4.7.1	Exception Handling	211
4.7.2	Assertions	217
4.8	Weiterführende Literatur	220

II Bausteine stabiler Java-Applikationen

221

5	Das Collections-Framework	223
5.1	Datenstrukturen	223
5.1.1	Wahl einer geeigneten Datenstruktur	224
5.1.2	Arrays	226
5.1.3	Das Interface <code>Collection</code>	228
5.1.4	Listen und das Interface <code>List</code>	233
5.1.5	Mengen und das Interface <code>Set</code>	239
5.1.6	Grundlagen von hashbasierten Containern	241
5.1.7	Grundlagen automatisch sortierender Container	250
5.1.8	Die Methoden <code>equals()</code> , <code>hashCode()</code> und <code>compareTo()</code> im Zusammenspiel	257
5.1.9	Konkrete Realisierungen von Mengen	259
5.1.10	Schlüssel-Wert-Abbildungen und das Interface <code>Map</code>	262
5.1.11	Erweiterungen am Beispiel der Klasse <code>HashMap</code>	269
5.1.12	Entscheidungshilfe zur Wahl von Datenstrukturen	272
5.2	Suchen, Sortieren und Filtern	273
5.2.1	Suchen	274
5.2.2	Sortieren mit Komparatoren	276
5.2.3	Filtern von Collections	282
5.3	Utility-Klassen und Hilfsmethoden	288
5.3.1	Nützliche Hilfsmethoden	289
5.3.2	Dekorierer <code>synchronized</code> , <code>unmodifiable</code> und <code>checked</code>	290
5.3.3	Vordefinierte Algorithmen	295
5.3.4	Design eines Zugriffsinterface	298
5.4	Probleme im Collections-Framework	302

5.4.1	Merkwürdigkeiten in Arrays	302
5.4.2	Probleme von <code>Stack</code> , <code>Queue</code> und <code>Deque</code>	304
5.5	Weiterführende Literatur	307
6	Applikationsbausteine	309
6.1	Einsatz von Bibliotheken am Beispiel	309
6.2	Wertebereichs- und Parameterprüfungen	314
6.2.1	Prüfung einfacher Wertebereiche und Wertemengen	314
6.2.2	Prüfung komplexerer Wertebereiche	318
6.3	Logging-Frameworks	326
6.3.1	Apache <code>log4j</code>	327
6.3.2	Tipps und Tricks zum Einsatz von Logging mit <code>log4j</code>	334
6.4	Utility-Klassen zur Dateibehandlung	342
6.4.1	Die Klasse <code>FileUtils</code>	342
6.4.2	Die Klasse <code>StreamUtils</code>	344
6.4.3	Implementierung von <code>StringStreams</code>	349
6.5	Konfigurationsparameter und -dateien	351
6.5.1	Einlesen von Kommandozeilenparametern	351
6.5.2	Verarbeitung von <code>Properties</code>	360
6.5.3	Die Klasse <code>Preferences</code>	366
6.5.4	Weitere Möglichkeiten zur Konfigurationsverwaltung	368
7	Multithreading	375
7.1	Threads und <code>Runnable</code> s	377
7.1.1	Definition der auszuführenden Aufgabe	377
7.1.2	Start, Ausführung und Ende von Threads	378
7.1.3	Lebenszyklus von Threads und Thread-Zustände	380
7.2	Zusammenarbeit von Threads	386
7.2.1	Konkurrierende Datenzugriffe	386
7.2.2	Locks, Monitore und kritische Bereiche	387
7.2.3	Deadlocks und Starvation	392
7.2.4	Kritische Bereiche und das Interface <code>Lock</code>	394
7.3	Kommunikation von Threads	398
7.3.1	Kommunikation mit Synchronisation	399
7.3.2	Kommunikation über die Methoden <code>wait()</code> , <code>notify()</code> und <code>notifyAll()</code>	402
7.3.3	Abstimmung von Threads	411
7.3.4	Unerwartete <code>IllegalMonitorStateException</code> s	415
7.4	Das Java-Memory-Modell	416
7.4.1	Sichtbarkeit	417
7.4.2	Atomarität	418
7.4.3	Reorderings	420
7.5	Besonderheiten bei Threads	423

7.5.1	Verschiedene Arten von Threads	423
7.5.2	Exceptions in Threads	425
7.5.3	Sicheres Beenden von Threads	426
7.5.4	Zeitgesteuerte Ausführung	430
7.6	Die Concurrency Utilities	433
7.6.1	Concurrent Collections	434
7.6.2	Das Executor-Framework	443
7.7	Weiterführende Literatur	454
8	Fortgeschrittene Java-Themen	455
8.1	Crashkurs Reflection	455
8.1.1	Grundlagen	457
8.1.2	Zugriff auf Methoden und Attribute	460
8.1.3	Spezialfälle	465
8.2	Serialisierung	468
8.2.1	Implementieren der Serialisierung	468
8.2.2	Die Serialisierung anpassen	473
8.2.3	Versionsverwaltung der Serialisierung	477
8.2.4	Optimierung der Serialisierung	480
8.3	Objektkopien und das Interface <code>Cloneable</code>	485
8.3.1	Das Interface <code>Cloneable</code>	486
8.3.2	Alternativen zur Methode <code>clone()</code>	495
8.4	Internationalisierung	497
8.4.1	Grundlagen	498
8.4.2	Die Klasse <code>Locale</code>	499
8.4.3	Die Klasse <code>PropertyResourceBundle</code>	502
8.4.4	Formatierte Ein- und Ausgabe	506
8.4.5	Zahlen und die Klasse <code>NumberFormat</code>	507
8.4.6	Datumswerte und die Klasse <code>DateFormat</code>	510
8.4.7	Textmeldungen und die Klasse <code>MessageFormat</code>	515
8.5	Programmbausteine zur Internationalisierung	517
8.5.1	Unterstützung mehrerer Datumsformate	517
8.5.2	Nutzung mehrerer Sprachdateien	522
8.6	Multithreading und Swing	532
8.6.1	Crashkurs Event Handling in Swing	532
8.6.2	Ausführen von Aktionen	534
8.6.3	Die Klasse <code>SwingWorker</code>	538
8.7	Garbage Collection	542
8.7.1	Einflussfaktoren auf die Garbage Collection	542
8.7.2	Algorithmen zur Garbage Collection	546
8.7.3	Optimierungen der Garbage Collection	548
8.7.4	Memory Leaks: Gibt es die auch in Java?!	549
8.7.5	Objekterstörung und <code>finalize()</code>	551

8.8	Weiterführende Literatur	553
9	Neuerungen in JDK 7	555
9.1	Erweiterungen der Sprache selbst	555
9.2	Erweiterungen des NIO in JDK 7	563
9.2.1	Dateibehandlung in JDK 7	563
9.2.2	Asynchronous I/O	569
9.3	Multithreading	570
9.4	Neuerungen in AWT und Swing	573
9.5	Collections	574
9.6	Der Garbage Collector »G1«	575
9.7	Geplante Datumsverarbeitung mit dem neuen Datums-API	576

III Fallstricke und Lösungen im Praxisalltag 581

10	Bad Smells	583
10.1	Programmdesign	585
10.1.1	Bad Smell: Verwenden von Magic Numbers	585
10.1.2	Bad Smell: Konstanten in Interfaces definieren	586
10.1.3	Bad Smell: <code>System.exit()</code> mitten im Programm	589
10.1.4	Bad Smell: Zusammengehörende Konstanten nicht als Typ definiert	590
10.1.5	Bad Smell: Programmcode im Logging-Code	592
10.1.6	Bad Smell: Unvollständige Betrachtung aller Alternativen ..	593
10.1.7	Bad Smell: Unvollständige Änderungen nach Copy-Paste ..	594
10.1.8	Bad Smell: Casts auf unbekannte Subtypen	596
10.1.9	Bad Smell: Pre-/Post-Increment in komplexeren Statements ..	598
10.1.10	Bad Smell: Keine Klammern um Blöcke	600
10.1.11	Bad Smell: Variablendeklaration nicht im kleinstmöglichen Sichtbarkeitsbereich	602
10.1.12	Bad Smell: Mehrere aufeinanderfolgende Parameter gleichen Typs	603
10.1.13	Bad Smell: Grundloser Einsatz von Reflection	604
10.2	Klassendesign	606
10.2.1	Bad Smell: Unnötigerweise veränderliche Attribute	606
10.2.2	Bad Smell: Aufruf abstrakter Methoden im Konstruktor ...	608
10.2.3	Bad Smell: Herausgabe von <code>this</code> im Konstruktor	612
10.2.4	Bad Smell: Referenzierung von Subklassen in Basisklassen ..	613
10.2.5	Bad Smell: Mix abstrakter und konkreter Basisklassen ...	615
10.2.6	Bad Smell: Öffentlicher Defaultkonstruktor lediglich zum Zugriff auf Hilfsmethoden	617
10.3	Fehlerbehandlung und Exception Handling	619

10.3.1	Bad Smell: Unbehandelte Exception	619
10.3.2	Bad Smell: Unpassender Exception-Typ	620
10.3.3	Bad Smell: Exceptions zur Steuerung des Kontrollflusses	622
10.3.4	Bad Smell: Fangen der allgemeinsten Exception	623
10.3.5	Bad Smell: Rückgabe von <code>null</code> statt Exception im Fehlerfall	625
10.3.6	Bad Smell: Unbedachte Rückgabe von <code>null</code>	627
10.3.7	Bad Smell: Sonderbehandlung von Randfällen	629
10.3.8	Bad Smell: Keine Gültigkeitsprüfung von Eingabeparametern	630
10.3.9	Bad Smell: Fehlerhafte Fehlerbehandlung	632
10.3.10	Bad Smell: I/O ohne <code>finally</code> bzw. <code>finalize()</code>	634
10.3.11	Bad Smell: Resource Leaks durch Exceptions im Konstruktor	636
10.4	Häufige Fallstricke	640
10.5	Weiterführende Literatur	648
11	Refactorings	649
11.1	Das Standardvorgehen	657
11.2	Der Refactoring-Katalog	660
11.2.1	Reduziere die Sichtbarkeit von Attributen	660
11.2.2	Minimiere veränderliche Attribute	663
11.2.3	Reduziere die Sichtbarkeit von Methoden	667
11.2.4	Ersetze Mutator- durch Business-Methode	668
11.2.5	Minimiere Zustandsänderungen (Refactoring-Kombination)	669
11.2.6	Führe ein Interface ein	669
11.2.7	Aufspalten eines Interface	670
11.2.8	Einführen eines Read-only-Interface	671
11.2.9	Einführen eines Read-Write-Interface	671
11.2.10	Einführen von Convenience-Methoden	672
11.2.11	Einführen einer Zustandsprüfung	674
11.2.12	Überprüfung von Eingabeparametern	676
11.2.13	Trenne Informationsbeschaffung und -verarbeitung	680
11.2.14	Konstantensammlung in <code>enum</code> umwandeln	685
11.2.15	Entferne Exceptions zur Steuerung des Kontrollflusses ...	688
11.2.16	Umwandlung in Utility-Klasse mit statischen Hilfsmethoden	691
11.3	Weiterführende Literatur	694
12	Entwurfsmuster	695
12.1	Erzeugungsmuster	698
12.1.1	Erzeugungsmethode	698
12.1.2	Fabrikmethode (Factory method)	701
12.1.3	Erbauer (Builder)	704

12.1.4	Singleton	707
12.1.5	Prototyp (Prototype)	711
12.2	Strukturmuster	715
12.2.1	Fassade (Façade)	716
12.2.2	Adapter	718
12.2.3	Dekorierer (Decorator)	720
12.2.4	Kompositum (Composite)	723
12.3	Verhaltensmuster	727
12.3.1	Iterator	728
12.3.2	Null-Objekt (Null Object)	730
12.3.3	Schablonenmethode (Template method)	733
12.3.4	Strategie (Strategy)	737
12.3.5	Befehl (Command)	745
12.3.6	Proxy	752
12.3.7	Zuständigkeitskette (Chain of Responsibility)	754
12.3.8	Beobachter (Observer)	756
12.3.9	MVC-Architektur	765
12.4	Weiterführende Literatur	766

IV Qualitätssicherungsmaßnahmen

767

13	Programmierstil und Coding Conventions	769
13.1	Grundregeln eines guten Programmierstils	769
13.1.1	Keep It Human-Readable	770
13.1.2	Keep It Simple And Short	770
13.1.3	Keep It Natural	770
13.1.4	Keep It Clean	770
13.2	Die Psychologie beim Sourcecode-Layout	771
13.2.1	Faktor der Ähnlichkeit	771
13.2.2	Faktor der Nähe	772
13.3	Coding Conventions	774
13.3.1	Grundlegende Namens- und Formatierungsregeln	775
13.3.2	Namensgebung	778
13.3.3	Dokumentation	780
13.3.4	Programmdesign	782
13.3.5	Klassendesign	787
13.3.6	Parameterlisten	790
13.3.7	Logik und Kontrollfluss	792
13.4	Sourcecode-Überprüfung mit Tools	794
13.4.1	Metriken	795
13.4.2	Sourcecode-Überprüfung im Build-Prozess	799

14	Unit Tests	807
14.1	Überblick	807
14.1.1	Arten von Tests	807
14.1.2	Äußere vs. innere Qualität	810
14.1.3	Auswirkungen von Unit Tests auf die Qualität	811
14.2	Motivation für Unit Tests aus der Praxis	813
14.2.1	Unit Tests für Neuentwicklungen	813
14.2.2	Unit Tests und Legacy-Code	820
14.3	Fortgeschrittene Unit-Test-Techniken	831
14.3.1	Testen mit Stubs	831
14.3.2	Testen mit Mocks	833
14.3.3	Unit Tests von privaten Methoden	836
14.4	Unit Tests mit Threads und Timing	837
14.5	Nützliche Tools für Unit Tests	842
14.5.1	Hamcrest	842
14.5.2	Infinitest	846
14.5.3	Cobertura	847
14.6	Weiterführende Literatur	853
15	Codereviews	855
15.1	Definition	855
15.2	Probleme und Tipps zur Durchführung	857
15.3	Vorteile von Codereviews	859
15.4	Codereview-Tools	862
15.5	Codereview-Checkliste	864
16	Optimierungen	865
16.1	Grundlagen	866
16.1.1	Optimierungsebenen und Einflussfaktoren	867
16.1.2	Optimierungstechniken	868
16.1.3	CPU-bound-Optimierungsebenen am Beispiel	870
16.1.4	Messungen – Erkennen kritischer Bereiche	874
16.1.5	Abschätzungen mit der O-Notation	881
16.2	Einsatz geeigneter Datenstrukturen	884
16.2.1	Einfluss von Arrays und Listen	885
16.2.2	Optimierungen für Set und Map	889
16.2.3	API-Design Collection vs. Iterator	891
16.3	Lazy Initialization	892
16.3.1	Lazy Initialization am Beispiel	893
16.3.2	Konsequenzen des Einsatzes der Lazy Initialization	896
16.3.3	Lazy Initialization mithilfe des PROXY-Musters	898
16.4	Optimierungen am Beispiel	900
16.5	I/O-bound-Optimierungen	908

16.5.1	Technik – Wahl passender Strategien	908
16.5.2	Technik – Caching und Pooling	911
16.5.3	Technik – Vermeidung unnötiger Aktionen	912
16.6	Memory-bound-Optimierungen	915
16.6.1	Technik – Wahl passender Strategien	915
16.6.2	Technik – Caching und Pooling	918
16.6.3	Optimierungen der Stringverarbeitung	923
16.6.4	Technik – Vermeidung unnötiger Aktionen	926
16.7	CPU-bound-Optimierungen	928
16.7.1	Technik – Wahl passender Strategien	929
16.7.2	Technik – Caching und Pooling	929
16.7.3	Technik – Vermeidung unnötiger Aktionen	930
16.8	Weiterführende Literatur	932

V Anhang **933**

A	Einführung in die UML	935
A.1	Die UML im Überblick	935
A.2	Strukturdiagramme – statische Modelle	939
A.2.1	Klassendiagramme	939
A.2.2	Objektdiagramme	943
A.2.3	Komponentendiagramme	943
A.2.4	Paketdiagramme	944
A.3	Verhaltensdiagramme – dynamische Modelle	945
A.3.1	Anwendungsfalldiagramme	945
A.3.2	Sequenzdiagramme	946
A.3.3	Kommunikationsdiagramme	950
A.3.4	Zustandsdiagramme	951
A.3.5	Aktivitätsdiagramme	953
A.4	Weiterführende Literatur	954
B	Überblick über den Softwareentwicklungsprozess	955
B.1	Vorgehensmodelle	955
B.1.1	Aufgaben und Phasen beim Softwareentwurf	955
B.1.2	Wasserfallmodell und V-Modell	956
B.1.3	Extreme Programming (XP)	958
B.1.4	Test-Driven Development (TDD)	960
B.1.5	Diskussion	961
	Literaturverzeichnis	963
	Stichwortverzeichnis	966