

Auf einen Blick

1	Java ist auch eine Sprache	47
2	Imperative Sprachkonzepte	97
3	Klassen und Objekte	223
4	Arrays und ihre Anwendungen	267
5	Der Umgang mit Zeichenketten	315
6	Eigene Klassen schreiben	395
7	Objektorientierte Beziehungsfragen	457
8	Ausnahmen müssen sein	553
9	Geschachtelte Typen	623
10	Besondere Typen der Java SE	643
11	Generics<T>	727
12	Lambda-Ausdrücke und funktionale Programmierung	785
13	Architektur, Design und angewandte Objektorientierung	857
14	Java Platform Module System	867
15	Die Klassenbibliothek	889
16	Einführung in die nebenläufige Programmierung	935
17	Einführung in Datenstrukturen und Algorithmen	977
18	Einführung in grafische Oberflächen	1033
19	Einführung in Dateien und Datenströme	1059
20	Einführung ins Datenbankmanagement mit JDBC	1097
21	Bits und Bytes, Mathematisches und Geld	1103
22	Testen mit JUnit	1163
23	Die Werkzeuge des JDK	1187

Inhalt

Materialien zum Buch	30
Vorwort	31

1

Java ist auch eine Sprache

47

1.1	Historischer Hintergrund	47
1.2	Warum Java populär ist – die zentralen Eigenschaften	49
1.2.1	Bytecode	50
1.2.2	Ausführung des Bytecodes durch eine virtuelle Maschine	50
1.2.3	Plattformunabhängigkeit	51
1.2.4	Java als Sprache, Laufzeitumgebung und Standardbibliothek	51
1.2.5	Objektorientierung in Java	52
1.2.6	Java ist verbreitet und bekannt	53
1.2.7	Java ist schnell – Optimierung und Just-in-time-Compilation	53
1.2.8	Das Java-Security-Modell	55
1.2.9	Zeiger und Referenzen	56
1.2.10	Bring den Müll raus, Garbage-Collector!	57
1.2.11	Ausnahmebehandlung	58
1.2.12	Das Angebot an Bibliotheken und Werkzeugen	59
1.2.13	Vergleichbar einfache Syntax	59
1.2.14	Java ist Open Source	61
1.2.15	Wofür sich Java weniger eignet	62
1.3	Java im Vergleich zu anderen Sprachen *	63
1.3.1	Java und C(++)	63
1.3.2	Java und JavaScript	64
1.3.3	Ein Wort zu Microsoft, Java und zu J++	64
1.3.4	Java und C#/.NET	65
1.4	Weiterentwicklung und Verluste	66
1.4.1	Die Entwicklung von Java und seine Zukunftsaussichten	66
1.4.2	Features, Enhancements (Erweiterungen) und ein JSR	67
1.4.3	Applets	68
1.4.4	JavaFX	69
1.5	Java-Plattformen: Java SE, Jakarta EE, Java ME, Java Card	69
1.5.1	Die Java SE-Plattform	70
1.5.2	Java ME: Java für die Kleinen	72

- 1.5.3 Java für die ganz, ganz Kleinen 73
 - 1.5.4 Java für die Großen: Jakarta EE (ehemals Java EE) 73
 - 1.5.5 Echtzeit-Java (Real-time Java) 74
- 1.6 Java SE-Implementierungen 75
 - 1.6.1 OpenJDK 75
 - 1.6.2 Oracle JDK 77
- 1.7 AdoptOpenJDK installieren 79
 - 1.7.1 AdoptOpenJDK unter Windows installieren 80
- 1.8 Das erste Programm compilieren und testen 82
 - 1.8.1 Ein Quadratzahlen-Programm 82
 - 1.8.2 Der Compilerlauf 83
 - 1.8.3 Die Laufzeitumgebung 84
 - 1.8.4 Häufige Compiler- und Interpreter-Probleme 85
- 1.9 Entwicklungsumgebungen 85
 - 1.9.1 Eclipse IDE 86
 - 1.9.2 IntelliJ IDEA 94
 - 1.9.3 NetBeans 96
- 1.10 Zum Weiterlesen 96

2 Imperative Sprachkonzepte97

- 2.1 Elemente der Programmiersprache Java 97
 - 2.1.1 Token 97
 - 2.1.2 Textkodierung durch Unicode-Zeichen 98
 - 2.1.3 Bezeichner 98
 - 2.1.4 Literale 101
 - 2.1.5 (Reservierte) Schlüsselwörter 101
 - 2.1.6 Zusammenfassung der lexikalischen Analyse 102
 - 2.1.7 Kommentare 103
- 2.2 Von der Klasse zur Anweisung 105
 - 2.2.1 Was sind Anweisungen? 105
 - 2.2.2 Klassendeklaration 106
 - 2.2.3 Die Reise beginnt am main(String[]) 107
 - 2.2.4 Der erste Methodenaufruf: println(...) 107
 - 2.2.5 Atomare Anweisungen und Anweisungssequenzen 109
 - 2.2.6 Mehr zu print(...), println(...) und printf(...) für Bildschirmausgaben 109
 - 2.2.7 Die API-Dokumentation 111
 - 2.2.8 Ausdrücke 112

2.2.9	Ausdrucksanweisung	113
2.2.10	Erster Einblick in die Objektorientierung	113
2.2.11	Modifizierer	114
2.2.12	Gruppieren von Anweisungen mit Blöcken	115
2.3	Datentypen, Typisierung, Variablen und Zuweisungen	116
2.3.1	Primitive Datentypen im Überblick	118
2.3.2	Variablendeklarationen	121
2.3.3	Automatisches Feststellen der Typen mit var	124
2.3.4	Finale Variablen und der Modifizierer final	125
2.3.5	Konsoleneingaben	126
2.3.6	Fließkommazahlen mit den Datentypen float und double	128
2.3.7	Ganzzahlige Datentypen	130
2.3.8	Wahrheitswerte	131
2.3.9	Unterstriche in Zahlen	132
2.3.10	Alphanumerische Zeichen	133
2.3.11	Gute Namen, schlechte Namen	133
2.3.12	Keine automatische Initialisierung von lokalen Variablen	134
2.4	Ausdrücke, Operanden und Operatoren	135
2.4.1	Zuweisungsoperator	135
2.4.2	Arithmetische Operatoren	137
2.4.3	Unäres Minus und Plus	140
2.4.4	Präfix- oder Postfix-Inkrement und -Dekrement	141
2.4.5	Zuweisung mit Operation (Verbundoperator)	144
2.4.6	Die relationalen Operatoren und die Gleichheitsoperatoren	145
2.4.7	Logische Operatoren: Nicht, Und, Oder, XOR	147
2.4.8	Kurzschluss-Operatoren	148
2.4.9	Der Rang der Operatoren in der Auswertungsreihenfolge	149
2.4.10	Die Typumwandlung (das Casting)	152
2.4.11	Überladenes Plus für Strings	158
2.4.12	Operator vermisst *	159
2.5	Bedingte Anweisungen oder Fallunterscheidungen	160
2.5.1	Verzweigung mit der if-Anweisung	160
2.5.2	Die Alternative mit einer if-else-Anweisung wählen	162
2.5.3	Der Bedingungsoperator	166
2.5.4	Die switch-Anweisung bietet die Alternative	169
2.5.5	Switch-Ausdrücke	176
2.6	Immer das Gleiche mit den Schleifen	179
2.6.1	Die while-Schleife	179
2.6.2	Die do-while-Schleife	181
2.6.3	Die for-Schleife	183

2.6.4	Schleifenbedingungen und Vergleiche mit == *	187
2.6.5	Schleifenabbruch mit break und zurück zum Test mit continue	189
2.6.6	break und continue mit Marken *	192
2.7	Methoden einer Klasse	196
2.7.1	Bestandteile einer Methode	196
2.7.2	Signatur-Beschreibung in der Java-API	198
2.7.3	Aufruf einer Methode	199
2.7.4	Methoden ohne Parameter deklarieren	200
2.7.5	Statische Methoden (Klassenmethoden)	201
2.7.6	Parameter, Argument und Wertübergabe	202
2.7.7	Methoden vorzeitig mit return beenden	204
2.7.8	Nicht erreichbarer Quellcode bei Methoden *	205
2.7.9	Methoden mit Rückgaben	205
2.7.10	Methoden überladen	210
2.7.11	Gültigkeitsbereich	213
2.7.12	Vorgegebener Wert für nicht aufgeführte Argumente *	214
2.7.13	Rekursive Methoden *	215
2.7.14	Die Türme von Hanoi *	220
2.8	Zum Weiterlesen	222
3	Klassen und Objekte	223
3.1	Objektorientierte Programmierung (OOP)	223
3.1.1	Warum überhaupt OOP?	223
3.1.2	Denk ich an Java, denk ich an Wiederverwendbarkeit	224
3.2	Eigenschaften einer Klasse	225
3.2.1	Klassenarbeit mit Point	226
3.3	Natürlich modellieren mit der UML (Unified Modeling Language) *	226
3.3.1	Hintergrund und Geschichte der UML *	227
3.3.2	Wichtige Diagrammtypen der UML *	228
3.3.3	UML-Werkzeuge *	229
3.4	Neue Objekte erzeugen	230
3.4.1	Ein Exemplar einer Klasse mit dem Schlüsselwort new anlegen	231
3.4.2	Deklarieren von Referenzvariablen	231
3.4.3	Jetzt mach mal 'nen Punkt: Zugriff auf Objektattribute und -methoden	232
3.4.4	Der Zusammenhang von new, Heap und Garbage-Collector	237
3.4.5	Überblick über Point-Methoden	238
3.4.6	Konstruktoren nutzen	241

3.5	ZZZZZnake	242
3.6	Pakete schnüren, Importe und Compilationseinheiten	245
3.6.1	Java-Pakete	245
3.6.2	Pakete der Standardbibliothek	246
3.6.3	Volle Qualifizierung und import-Deklaration	246
3.6.4	Mit import p1.p2.* alle Typen eines Pakets erreichen	248
3.6.5	Hierarchische Strukturen über Pakete und die Spiegelung im Dateisystem ...	249
3.6.6	Die package-Deklaration	249
3.6.7	Unbenanntes Paket (default package)	251
3.6.8	Compilationseinheit (Compilation Unit)	252
3.6.9	Statischer Import *	252
3.7	Mit Referenzen arbeiten, Vielfalt, Identität, Gleichwertigkeit	253
3.7.1	null-Referenz und die Frage der Philosophie	254
3.7.2	Alles auf null? Referenzen testen	256
3.7.3	Zuweisungen bei Referenzen	257
3.7.4	Methoden mit Referenztypen als Parametern	258
3.7.5	Identität von Objekten	263
3.7.6	Gleichwertigkeit und die Methode equals(...)	263
3.8	Zum Weiterlesen	265

4 Arrays und ihre Anwendungen 267

4.1	Einfache Feldarbeit	267
4.1.1	Grundbestandteile	267
4.1.2	Deklaration von Array-Variablen	268
4.1.3	Array-Objekte mit new erzeugen	269
4.1.4	Arrays mit { Inhalt }	270
4.1.5	Die Länge eines Arrays über das Attribut length auslesen	271
4.1.6	Zugriff auf die Elemente über den Index	272
4.1.7	Typische Array-Fehler	273
4.1.8	Arrays an Methoden übergeben	275
4.1.9	Mehrere Rückgabewerte *	276
4.1.10	Vorinitialisierte Arrays	276
4.2	Die erweiterte for-Schleife	278
4.3	Methode mit variabler Argumentanzahl (Varargs)	282
4.3.1	System.out.printf(...) nimmt eine beliebige Anzahl von Argumenten an	282
4.3.2	Durchschnitt finden von variablen Argumenten	283
4.3.3	Varargs-Designtipps *	284

- 4.4 Mehrdimensionale Arrays * 285
 - 4.4.1 Nichtrechteckige Arrays * 288
- 4.5 Bibliotheksunterstützung von Arrays 291
 - 4.5.1 Klonen kann sich lohnen – Arrays vermehren 291
 - 4.5.2 Warum »können« Arrays so wenig? 292
 - 4.5.3 Array-Inhalte kopieren 292
- 4.6 Die Klasse Arrays zum Vergleichen, Füllen, Suchen und Sortieren nutzen 293
 - 4.6.1 Eine lange Schlange 307
- 4.7 Der Einstiegspunkt für das Laufzeitsystem: main(...) 310
 - 4.7.1 Korrekte Deklaration der Startmethode 310
 - 4.7.2 Kommandozeilenargumente verarbeiten 311
 - 4.7.3 Der Rückgabotyp von main(...) und System.exit(int) * 311
- 4.8 Zum Weiterlesen 313

5 Der Umgang mit Zeichenketten 315

- 5.1 Von ASCII über ISO-8859-1 zu Unicode 315
 - 5.1.1 ASCII 315
 - 5.1.2 ISO/IEC 8859-1 316
 - 5.1.3 Unicode 317
 - 5.1.4 Unicode-Zeichenkodierung 319
 - 5.1.5 Escape-Sequenzen/Fluchtsymbole 320
 - 5.1.6 Schreibweise für Unicode-Zeichen und Unicode-Escapes 321
 - 5.1.7 Java-Versionen gehen mit dem Unicode-Standard Hand in Hand * 323
- 5.2 Datentypen für Zeichen und Zeichenfolgen 324
- 5.3 Die Character-Klasse 325
 - 5.3.1 Ist das so? 325
 - 5.3.2 Zeichen in Großbuchstaben/Kleinbuchstaben konvertieren 327
 - 5.3.3 Vom Zeichen zum String 328
 - 5.3.4 Von char in int: vom Zeichen zur Zahl * 329
- 5.4 Zeichenfolgen 330
- 5.5 Die Klasse String und ihre Methoden 332
 - 5.5.1 String-Literale als String-Objekte für konstante Zeichenketten 333
 - 5.5.2 Konkatenation mit + 333
 - 5.5.3 String-Länge und Test auf Leer-String 333
 - 5.5.4 Zugriff auf ein bestimmtes Zeichen mit charAt(int) 335
 - 5.5.5 Nach enthaltenen Zeichen und Zeichenfolgen suchen 336

5.5.6	Das Hangman-Spiel	339
5.5.7	Gut, dass wir verglichen haben	340
5.5.8	String-Teile extrahieren	344
5.5.9	Strings anhängen, zusammenfügen, Groß-/Kleinschreibung und Weißraum	348
5.5.10	Gesucht, gefunden, ersetzt	352
5.5.11	String-Objekte mit Konstruktoren und aus Wiederholungen erzeugen *	354
5.6	Veränderbare Zeichenketten mit StringBuilder und StringBuffer	358
5.6.1	Anlegen von StringBuilder-Objekten	359
5.6.2	StringBuilder in andere Zeichenkettenformate konvertieren	360
5.6.3	Zeichen(folgen) erfragen	360
5.6.4	Daten anhängen	361
5.6.5	Zeichen(folgen) setzen, löschen und umdrehen	363
5.6.6	Länge und Kapazität eines StringBuilder-Objekts *	365
5.6.7	Vergleich von StringBuilder-Exemplaren und Strings mit StringBuilder	366
5.6.8	hashCode() bei StringBuilder *	368
5.7	CharSequence als Basistyp	368
5.8	Konvertieren zwischen Primitiven und Strings	371
5.8.1	Unterschiedliche Typen in String-Repräsentationen konvertieren	371
5.8.2	String-Inhalt in einen primitiven Wert konvertieren	373
5.8.3	String-Repräsentation im Format Binär, Hex und Oktal *	375
5.8.4	parseXXX(...)- und printXXX()-Methoden in DatatypeConverter *	379
5.9	Strings zusammenhängen (konkatenieren)	380
5.9.1	Strings mit StringJoiner zusammenhängen	381
5.10	Zerlegen von Zeichenketten	382
5.10.1	Splitten von Zeichenketten mit split(...)	383
5.10.2	Yes we can, yes we scan – die Klasse Scanner	384
5.11	Ausgaben formatieren	388
5.11.1	Formatieren und Ausgeben mit format()	388
5.12	Zum Weiterlesen	394

6 Eigene Klassen schreiben 395

6.1	Eigene Klassen mit Eigenschaften deklarieren	395
6.1.1	Attribute deklarieren	396
6.1.2	Methoden deklarieren	398
6.1.3	Verdeckte (shadowed) Variablen	402
6.1.4	Die this-Referenz	403

6.2	Privatsphäre und Sichtbarkeit	407
6.2.1	Für die Öffentlichkeit: public	407
6.2.2	Kein Public Viewing – Passwörter sind privat	408
6.2.3	Wieso nicht freie Methoden und Variablen für alle?	409
6.2.4	Privat ist nicht ganz privat: Es kommt darauf an, wer's sieht *	410
6.2.5	Zugriffsmethoden für Attribute deklarieren	410
6.2.6	Setter und Getter nach der JavaBeans-Spezifikation	411
6.2.7	Paketsichtbar	413
6.2.8	Zusammenfassung zur Sichtbarkeit	415
6.3	Eine für alle – statische Methoden und statische Attribute	417
6.3.1	Warum statische Eigenschaften sinnvoll sind	418
6.3.2	Statische Eigenschaften mit static	419
6.3.3	Statische Eigenschaften über Referenzen nutzen? *	420
6.3.4	Warum die Groß- und Kleinschreibung wichtig ist *	421
6.3.5	Statische Variablen zum Datenaustausch *	422
6.3.6	Statische Eigenschaften und Objekteigenschaften *	423
6.4	Konstanten und Aufzählungen	424
6.4.1	Konstanten über statische finale Variablen	425
6.4.2	Typunsichere Aufzählungen	426
6.4.3	Aufzählungstypen: typsichere Aufzählungen mit enum	428
6.5	Objekte anlegen und zerstören	432
6.5.1	Konstruktoren schreiben	432
6.5.2	Verwandtschaft von Methode und Konstruktor	434
6.5.3	Der Standard-Konstruktor (default constructor)	435
6.5.4	Parametrisierte und überladene Konstruktoren	436
6.5.5	Copy-Konstruktor	438
6.5.6	Einen anderen Konstruktor der gleichen Klasse mit this(...) aufrufen	440
6.5.7	Immutable-Objekte und Wither-Methoden	443
6.5.8	Ihr fehlt uns nicht – der Garbage-Collector	444
6.6	Klassen- und Objektinitialisierung *	446
6.6.1	Initialisierung von Objektvariablen	446
6.6.2	Statische Blöcke als Klasseninitialisierer	448
6.6.3	Initialisierung von Klassenvariablen	449
6.6.4	Eincompilierte Belegungen der Klassenvariablen	450
6.6.5	Exemplarinitialisierer (Instanzinitialisierer)	451
6.6.6	Finale Werte im Konstruktor und in statischen Blöcken setzen	454
6.7	Zum Weiterlesen	456

7	Objektorientierte Beziehungsfragen	457
7.1	Assoziationen zwischen Objekten	457
7.1.1	Unidirektionale 1:1-Beziehung	458
7.1.2	Zwei Freunde müsst ihr werden – bidirektionale 1:1-Beziehungen	459
7.1.3	Unidirektionale 1:n-Beziehung	460
7.2	Vererbung	465
7.2.1	Vererbung in Java	465
7.2.2	Spielobjekte modellieren	466
7.2.3	Die implizite Basisklasse java.lang.Object	468
7.2.4	Einfach- und Mehrfachvererbung *	469
7.2.5	Sehen Kinder alles? Die Sichtbarkeit protected	469
7.2.6	Konstruktoren in der Vererbung und super(...)	470
7.3	Typen in Hierarchien	476
7.3.1	Automatische und explizite Typumwandlung	476
7.3.2	Das Substitutionsprinzip	479
7.3.3	Typen mit dem instanceof-Operator testen	481
7.4	Methoden überschreiben	483
7.4.1	Methoden in Unterklassen mit neuem Verhalten ausstatten	483
7.4.2	Mit super an die Eltern	487
7.4.3	Finale Klassen und finale Methoden	490
7.4.4	Kovariante Rückgabetypen	491
7.4.5	Array-Typen und Kovarianz *	493
7.5	Drum prüfe, wer sich dynamisch bindet	494
7.5.1	Gebunden an toString()	494
7.5.2	Implementierung von System.out.println(Object)	496
7.5.3	Nicht dynamisch gebunden bei privaten, statischen und finalen Methoden	497
7.5.4	Dynamisch gebunden auch bei Konstruktoraufrufen *	498
7.5.5	Eine letzte Spielerei mit Javas dynamischer Bindung und überdeckten Attributen *	500
7.6	Abstrakte Klassen und abstrakte Methoden	502
7.6.1	Abstrakte Klassen	502
7.6.2	Abstrakte Methoden	504
7.7	Schnittstellen	510
7.7.1	Schnittstellen sind neue Typen	510
7.7.2	Schnittstellen deklarieren	511
7.7.3	Abstrakte Methoden in Schnittstellen	511
7.7.4	Implementieren von Schnittstellen	512
7.7.5	Ein Polymorphie-Beispiel mit Schnittstellen	514

7.7.6	Die Mehrfachvererbung bei Schnittstellen	516
7.7.7	Keine Kollisionsgefahr bei Mehrfachvererbung *	520
7.7.8	Erweitern von Interfaces – Subinterfaces	521
7.7.9	Konstantendeklarationen bei Schnittstellen	522
7.7.10	Nachträgliches Implementieren von Schnittstellen *	524
7.7.11	Statische ausprogrammierte Methoden in Schnittstellen	525
7.7.12	Erweitern und Ändern von Schnittstellen	527
7.7.13	Default-Methoden	529
7.7.14	Erweiterte Schnittstellen deklarieren und nutzen	530
7.7.15	Öffentliche und private Schnittstellenmethoden	533
7.7.16	Erweiterte Schnittstellen, Mehrfachvererbung und Mehrdeutigkeiten *	533
7.7.17	Bausteine bilden mit Default-Methoden *	537
7.7.18	Initialisierung von Schnittstellenkonstanten *	544
7.7.19	Markierungsschnittstellen *	547
7.7.20	(Abstrakte) Klassen und Schnittstellen im Vergleich	548
7.8	SOLiDe Modellierung	549
7.8.1	DRY, KISS und YAGNI	549
7.8.2	SOLID	549
7.8.3	Sei nicht STUPID	551
7.9	Zum Weiterlesen	552

8

Ausnahmen müssen sein

553

8.1	Problembereiche einzäunen	553
8.1.1	Exceptions in Java mit try und catch	554
8.1.2	Geprüfte und ungeprüfte Ausnahmen	554
8.2	Geprüfte Ausnahmen	555
8.2.1	Letzte ausgeführte Java-Programme loggen	555
8.2.2	try-catch-Behandlung	556
8.2.3	throws im Methodenkopf angeben	557
8.3	Ungeprüfte Ausnahmen (RuntimeException)	558
8.3.1	Eine NumberFormatException fliegt	559
8.3.2	Bekannte RuntimeException-Klassen	561
8.3.3	Kann man abfangen, muss man aber nicht	562
8.4	Gut gefangen	562
8.4.1	Bitte nicht schlucken – leere catch-Blöcke	562
8.4.2	Wiederholung abgebrochener Bereiche *	563
8.4.3	Mehrere Ausnahmen auffangen	563

8.4.4	Ablauf einer Ausnahmesituation	566
8.4.5	Abschlussbehandlung mit finally	567
8.5	Die Klassenhierarchie der Ausnahmen	572
8.5.1	Eigenschaften des Exception-Objekts	572
8.5.2	Basistyp Throwable	573
8.5.3	Die Exception-Hierarchie	574
8.5.4	Oberausnahmen auffangen	575
8.5.5	Schon gefangen?	577
8.5.6	Alles geht als Exception durch	578
8.5.7	Zusammenfassen gleicher catch-Blöcke mit dem multi-catch	579
8.6	Auslösen eigener Exceptions	583
8.6.1	Mit throw Ausnahmen auslösen	583
8.6.2	Vorhandene Runtime-Ausnahmetypen kennen und nutzen	585
8.6.3	Parameter testen und gute Fehlermeldungen	588
8.6.4	Neue Exception-Klassen deklarieren	589
8.6.5	Eigene Ausnahmen als Unterklassen von Exception oder RuntimeException?	590
8.6.6	Ausnahmen abfangen und weiterleiten *	593
8.6.7	Aufruf-Stack von Ausnahmen verändern *	595
8.6.8	Präzises rethrow *	596
8.6.9	Geschachtelte Ausnahmen *	599
8.7	Automatisches Ressourcen-Management (try mit Ressourcen)	602
8.7.1	try mit Ressourcen	603
8.7.2	Die Schnittstelle AutoCloseable	604
8.7.3	Mehrere Ressourcen nutzen	606
8.7.4	try mit Ressourcen auf null-Ressourcen	607
8.7.5	Ausnahmen vom close()	608
8.7.6	Unterdrückte Ausnahmen *	609
8.8	Besonderheiten bei der Ausnahmebehandlung *	612
8.8.1	Rückgabewerte bei ausgelösten Ausnahmen	612
8.8.2	Ausnahmen und Rückgaben verschwinden – das Duo return und finally	613
8.8.3	throws bei überschriebenen Methoden	614
8.8.4	Nicht erreichbare catch-Klauseln	616
8.9	Harte Fehler – Error *	617
8.10	Assertions *	618
8.10.1	Assertions in eigenen Programmen nutzen	619
8.10.2	Assertions aktivieren und Laufzeit-Errors	619
8.10.3	Assertions feiner aktivieren oder deaktivieren	622
8.11	Zum Weiterlesen	622

9	Geschachtelte Typen	623
9.1	Geschachtelte Klassen, Schnittstellen und Aufzählungen	623
9.2	Statische geschachtelte Typen	624
9.3	Nichtstatische geschachtelte Typen	626
9.3.1	Exemplare innerer Klassen erzeugen	626
9.3.2	Die this-Referenz	627
9.3.3	Vom Compiler generierte Klassendateien *	628
9.3.4	Erlaubte Modifizierer bei äußeren und inneren Klassen	629
9.4	Lokale Klassen	629
9.4.1	Beispiel mit eigener Klassendeklaration	630
9.4.2	Lokale Klasse für einen Timer nutzen	631
9.5	Anonyme innere Klassen	631
9.5.1	Nutzung einer anonymen inneren Klasse für den Timer	632
9.5.2	Umsetzung innerer anonymer Klassen *	633
9.5.3	Konstruktoren innerer anonymer Klassen	634
9.6	Zugriff auf lokale Variablen aus lokalen und anonymen Klassen *	636
9.7	this in Unterklassen *	637
9.7.1	Geschachtelte Klassen greifen auf private Eigenschaften zu	638
9.8	Nester	640
9.9	Zum Weiterlesen	641
10	Besondere Typen der Java SE	643
10.1	Object ist die Mutter aller Klassen	644
10.1.1	Klassenobjekte	644
10.1.2	Objektidentifikation mit toString()	645
10.1.3	Objektgleichwertigkeit mit equals(...) und Identität	647
10.1.4	Klonen eines Objekts mit clone() *	653
10.1.5	Hashwerte über hashCode() liefern *	658
10.1.6	System.identityHashCode(...) und das Problem der nicht eindeutigen Objektverweise *	665
10.1.7	Aufräumen mit finalize() *	666
10.1.8	Synchronisation *	669
10.2	Schwache Referenzen und Cleaner	669

10.3	Die Utility-Klasse java.util.Objects	670
10.3.1	Eingebaute null-Tests für equals(...)/hashCode()	670
10.3.2	Objects.toString(...)	671
10.3.3	null-Prüfungen mit eingebauter Ausnahmebehandlung	672
10.3.4	Tests auf null	673
10.3.5	Indexbezogene Programargumente auf Korrektheit prüfen	673
10.4	Vergleichen von Objekten und Ordnung herstellen	674
10.4.1	Natürlich geordnet oder nicht?	674
10.4.2	compareXXX()-Methode der Schnittstellen Comparable und Comparator	675
10.4.3	Rückgabewerte kodieren die Ordnung	676
10.4.4	Beispiel-Comparator: den kleinsten Raum einer Sammlung finden	677
10.4.5	Tipps für Comparator und Comparable-Implementierungen	679
10.4.6	Statische und Default-Methoden in Comparator	680
10.5	Wrapper-Klassen und Autoboxing	683
10.5.1	Wrapper-Objekte erzeugen	685
10.5.2	Konvertierungen in eine String-Repräsentation	686
10.5.3	Von einer String-Repräsentation parsen	687
10.5.4	Die Basisklasse Number für numerische Wrapper-Objekte	688
10.5.5	Vergleiche durchführen mit compareXXX(...), compareTo(...), equals(...) und Hashwerten	690
10.5.6	Statische Reduzierungsmethoden in Wrapper-Klassen	693
10.5.7	Konstanten für die Größe eines primitiven Typs	693
10.5.8	Behandeln von vorzeichenlosen Zahlen *	694
10.5.9	Die Klasse Integer	696
10.5.10	Die Klassen Double und Float für Fließkommazahlen	697
10.5.11	Die Long-Klasse	697
10.5.12	Die Boolean-Klasse	697
10.5.13	Autoboxing: Boxing und Unboxing	699
10.6	Iterator, Iterable *	703
10.6.1	Die Schnittstelle Iterator	703
10.6.2	Wer den Iterator liefert	706
10.6.3	Die Schnittstelle Iterable	707
10.6.4	Erweitertes for und Iterable	707
10.6.5	Interne Iteration	707
10.6.6	Eine eigene Iterable implementieren *	708
10.7	Die Spezial-Oberklasse Enum	710
10.7.1	Methoden auf Enum-Objekten	710
10.7.2	Aufzählungen mit eigenen Methoden und Initialisierern *	714
10.7.3	enum mit eigenen Konstruktoren *	716

10.8	Annotationen in der Java SE	720
10.8.1	Orte für Annotationen	720
10.8.2	Annotationstypen aus java.lang	721
10.8.3	@Deprecated	722
10.8.4	Annotationen mit zusätzlichen Informationen	722
10.8.5	@SuppressWarnings	723
10.9	Zum Weiterlesen	726
 11 Generics<T>		727
11.1	Einführung in Java Generics	727
11.1.1	Mensch versus Maschine – Typprüfung des Compilers und der Laufzeitumgebung	727
11.1.2	Raketen	728
11.1.3	Generische Typen deklarieren	730
11.1.4	Generics nutzen	731
11.1.5	Diamonds are forever	734
11.1.6	Generische Schnittstellen	737
11.1.7	Generische Methoden/Konstruktoren und Typ-Inferenz	739
11.2	Umsetzen der Generics, Typlöschung und Raw-Types	743
11.2.1	Realisierungsmöglichkeiten	743
11.2.2	Typlöschung (Type Erasure)	743
11.2.3	Probleme der Typlöschung	745
11.2.4	Raw-Type	750
11.3	Die Typen über Bounds einschränken	752
11.3.1	Einfache Einschränkungen mit extends	753
11.3.2	Weitere Obertypen mit &	755
11.4	Typparameter in der throws-Klausel *	756
11.4.1	Deklaration einer Klasse mit Typvariable <E extends Exception>	756
11.4.2	Parametrisierter Typ bei Typvariable <E extends Exception>	756
11.5	Generics und Vererbung, Invarianz	759
11.5.1	Arrays sind kovariant	759
11.5.2	Generics sind nicht kovariant, sondern invariant	760
11.5.3	Wildcards mit ?	761
11.5.4	Bounded Wildcards	763
11.5.5	Bounded-Wildcard-Typen und Bounded-Typvariablen	767
11.5.6	Das LESS-Prinzip	769
11.5.7	Enum<E extends Enum<E>> *	772

11.6	Konsequenzen der Typlöschung: Typ-Token, Arrays und Brücken *	773
11.6.1	Typ-Token	774
11.6.2	Super-Type-Token	775
11.6.3	Generics und Arrays	777
11.6.4	Brückenmethoden	778
11.7	Zum Weiterlesen	783
12	Lambda-Ausdrücke und funktionale Programmierung	785
12.1	Funktionale Schnittstellen und Lambda-Ausdrücke	785
12.1.1	Klassen implementieren Schnittstellen	785
12.1.2	Lambda-Ausdrücke implementieren Schnittstellen	787
12.1.3	Funktionale Schnittstellen	788
12.1.4	Der Typ eines Lambda-Ausdrucks ergibt sich durch den Zieltyp	789
12.1.5	Annotation @FunctionalInterface	794
12.1.6	Syntax für Lambda-Ausdrücke	795
12.1.7	Die Umgebung der Lambda-Ausdrücke und Variablenzugriffe	800
12.1.8	Ausnahmen in Lambda-Ausdrücken	806
12.1.9	Klassen mit einer abstrakten Methode als funktionale Schnittstelle? *	810
12.2	Methodenreferenz	811
12.2.1	Motivation	811
12.2.2	Methodenreferenzen mit ::	812
12.2.3	Varianten von Methodenreferenzen	813
12.3	Konstruktorreferenz	815
12.3.1	Parameterlose und parametrisierte Konstruktoren	817
12.3.2	Nützliche vordefinierte Schnittstellen für Konstruktorreferenzen	818
12.4	Funktionale Programmierung	819
12.4.1	Code = Daten	819
12.4.2	Programmierparadigmen: imperativ oder deklarativ	820
12.4.3	Das Wesen der funktionalen Programmierung	821
12.4.4	Funktionale Programmierung und funktionale Programmiersprachen	824
12.4.5	Funktionen höherer Ordnung am Beispiel von Comparator	826
12.4.6	Lambda-Ausdrücke als Abbildungen bzw. Funktionen betrachten	827
12.5	Funktionale Schnittstellen aus dem java.util.function-Paket	828
12.5.1	Blöcke mit Code und die funktionale Schnittstelle Consumer	828
12.5.2	Supplier	830
12.5.3	Prädikate und java.util.function.Predicate	830
12.5.4	Funktionen über die funktionale Schnittstelle java.util.function.Function	832

12.5.5	Ein bisschen Bi ...	836
12.5.6	Funktionale Schnittstellen mit Primitiven	839
12.6	Optional ist keine Nullnummer	842
12.6.1	Einsatz von null	842
12.6.2	Der Optional-Typ	845
12.6.3	Primitive optionale Typen	847
12.6.4	Erst mal funktional mit Optional	849
12.6.5	Primitiv-Optionales mit speziellen OptionalXXX-Klassen	852
12.7	Was ist jetzt so funktional?	854
12.8	Zum Weiterlesen	856

13 Architektur, Design und angewandte Objektorientierung

857

13.1	Architektur, Design und Implementierung	857
13.2	Design-Patterns (Entwurfsmuster)	858
13.2.1	Motivation für Design-Patterns	858
13.2.2	Singleton	859
13.2.3	Fabrikmethoden	860
13.2.4	Das Beobachter-Pattern mit Listener realisieren	861
13.3	Zum Weiterlesen	865

14 Java Platform Module System

867

14.1	Klassenlader (Class Loader) und Modul-/Klassenpfad	867
14.1.1	Klassenladen auf Abruf	867
14.1.2	Klassenlader bei der Arbeit zusehen	868
14.1.3	JMOD-Dateien und JAR-Dateien	869
14.1.4	Woher die kleinen Klassen kommen: Die Suchorte und spezielle Klassenlader	870
14.1.5	Setzen des Modulpfades	870
14.2	Module entwickeln und einbinden	873
14.2.1	Wer sieht wen?	873
14.2.2	Plattform-Module und ein JMOD-Beispiel	874
14.2.3	Interne Plattformeigenschaften nutzen, --add-exports	875

14.2.4	Neue Module einbinden, --add-modules und --add-opens	877
14.2.5	Projektabhängigkeiten in Eclipse	878
14.2.6	Benannte Module und module-info.java	880
14.2.7	Automatische Module	884
14.2.8	Unbenanntes Modul	885
14.2.9	Lesbarkeit und Zugreifbarkeit	886
14.2.10	Modul-Migration	887
14.3	Zum Weiterlesen	888

15

Die Klassenbibliothek

889

15.1	Die Java-Klassenphilosophie	889
15.1.1	Modul, Paket, Typ	889
15.1.2	Übersicht über die Pakete der Standardbibliothek	892
15.2	Einfache Zeitmessung und Profiling *	896
15.3	Die Klasse Class	899
15.3.1	An ein Class-Objekt kommen	900
15.3.2	Eine Class ist ein Type	902
15.4	Klassenlader	903
15.4.1	Die Klasse java.lang.ClassLoader	904
15.5	Die Utility-Klassen System und Properties	904
15.5.1	Speicher der JVM	905
15.5.2	Anzahl der CPUs bzw. Kerne	906
15.5.3	Systemeigenschaften der Java-Umgebung	907
15.5.4	Eigene Properties von der Konsole aus setzen *	908
15.5.5	Zeilenumbruchzeichen, line.separator	911
15.5.6	Umgebungsvariablen des Betriebssystems	911
15.6	Sprachen der Länder	913
15.6.1	Sprachen in Regionen über Locale-Objekte	913
15.7	Wichtige Datum-Klassen im Überblick	917
15.7.1	Der 1.1.1970	918
15.7.2	System.currentTimeMillis()	918
15.7.3	Einfache Zeitumrechnungen durch TimeUnit	919
15.8	Date-Time-API	920
15.8.1	Menschenzeit und Maschinenzeit	922
15.8.2	Die Datumsklasse LocalDate	924

- 15.9 Logging mit Java** 925
 - 15.9.1 Logging-APIs 926
 - 15.9.2 Logging mit java.util.logging 926
- 15.10 Maven: Build-Management und Abhängigkeiten auflösen** 929
 - 15.10.1 Beispielprojekt in Eclipse mit Maven 929
 - 15.10.2 Properties hinzunehmen 930
 - 15.10.3 Dependency hinzunehmen 930
 - 15.10.4 Lokales und das Remote-Repository 932
 - 15.10.5 Lebenszyklus, Phasen und Maven-Plugins 932
 - 15.10.6 Archetypes 932
- 15.11 Zum Weiterlesen** 933

- 16 Einführung in die nebenläufige Programmierung** 935

- 16.1 Nebenläufigkeit und Parallelität** 935
 - 16.1.1 Multitasking, Prozesse und Threads 936
 - 16.1.2 Threads und Prozesse 936
 - 16.1.3 Wie nebenläufige Programme die Geschwindigkeit steigern können 937
 - 16.1.4 Was Java für Nebenläufigkeit alles bietet 939
- 16.2 Existierende Threads und neue Threads erzeugen** 939
 - 16.2.1 Main-Thread 940
 - 16.2.2 Wer bin ich? 940
 - 16.2.3 Die Schnittstelle Runnable implementieren 940
 - 16.2.4 Thread mit Runnable starten 942
 - 16.2.5 Runnable parametrisieren 943
 - 16.2.6 Die Klasse Thread erweitern 944
- 16.3 Thread-Eigenschaften und Zustände** 946
 - 16.3.1 Der Name eines Threads 947
 - 16.3.2 Die Zustände eines Threads * 947
 - 16.3.3 Schläfer gesucht 948
 - 16.3.4 Wann Threads fertig sind 950
 - 16.3.5 Einen Thread höflich mit Interrupt beenden 950
 - 16.3.6 Unbehandelte Ausnahmen, Thread-Ende und UncaughtExceptionHandler ... 953
 - 16.3.7 Der stop() von außen und die Rettung mit ThreadDeath * 954
 - 16.3.8 Ein Rendezvous mit join(...) * 956
 - 16.3.9 Arbeit niederlegen und wieder aufnehmen * 958
 - 16.3.10 Priorität * : 958

16.4 Der Ausführer (Executor) kommt	960
16.4.1 Die Schnittstelle Executor	960
16.4.2 Glücklich in der Gruppe – die Thread-Pools	962
16.4.3 Threads mit Rückgabe über Callable	964
16.4.4 Erinnerungen an die Zukunft – die Future-Rückgabe	966
16.4.5 Mehrere Callable-Objekte abarbeiten	969
16.4.6 CompletionService und ExecutorCompletionService	970
16.4.7 ScheduledExecutorService: wiederholende Aufgaben und Zeitsteuerungen	972
16.4.8 Asynchrones Programmieren mit CompletableFuture (CompletionStage)	972
16.5 Zum Weiterlesen	975
 17 Einführung in Datenstrukturen und Algorithmen	 977
17.1 Listen	977
17.1.1 Erstes Listen-Beispiel	978
17.1.2 Auswahlkriterium ArrayList oder LinkedList	979
17.1.3 Die Schnittstelle List	979
17.1.4 ArrayList	985
17.1.5 LinkedList	987
17.1.6 Der Array-Adapter Arrays.asList(...)	989
17.1.7 ListIterator *	991
17.1.8 toArray(...) von Collection verstehen – die Gefahr einer Falle erkennen	992
17.1.9 Primitive Elemente in Datenstrukturen verwalten	995
17.2 Mengen (Sets)	996
17.2.1 Ein erstes Mengen-Beispiel	997
17.2.2 Methoden der Schnittstelle Set	998
17.2.3 HashSet	1000
17.2.4 TreeSet – die sortierte Menge	1001
17.2.5 Die Schnittstellen NavigableSet und SortedSet	1003
17.2.6 LinkedHashSet	1005
17.3 Java Stream-API	1007
17.3.1 Deklaratives Programmieren	1007
17.3.2 Interne versus externe Iteration	1007
17.3.3 Was ist ein Stream?	1008
17.4 Einen Stream erzeugen	1009
17.4.1 Parallele oder sequenzielle Streams	1013
17.5 Terminale Operationen	1014
17.5.1 Die Anzahl der Elemente	1014

- 17.5.2 Und jetzt alle – forEachXXX(...) 1014
 - 17.5.3 Einzelne Elemente aus dem Strom holen 1015
 - 17.5.4 Existenz-Tests mit Prädikaten 1016
 - 17.5.5 Einen Strom auf sein kleinstes bzw. größtes Element reduzieren 1016
 - 17.5.6 Einen Strom mit eigenen Funktionen reduzieren 1017
 - 17.5.7 Ergebnisse in einen Container schreiben, Teil 1: collect(...) 1018
 - 17.5.8 Ergebnisse in einen Container schreiben, Teil 2: Collector und Collectors 1020
 - 17.5.9 Ergebnisse in einen Container schreiben, Teil 3: Gruppierungen 1022
 - 17.5.10 Stream-Elemente in ein Array oder einen Iterator übertragen 1024
- 17.6 Intermediäre Operationen 1025
 - 17.6.1 Element-Vorschau 1025
 - 17.6.2 Filtern von Elementen 1026
 - 17.6.3 Statusbehaftete intermediäre Operationen 1026
 - 17.6.4 Präfix-Operation 1028
 - 17.6.5 Abbildungen 1029
- 17.7 Zum Weiterlesen 1031

18 Einführung in grafische Oberflächen

1033

- 18.1 GUI-Frameworks 1033
 - 18.1.1 Kommandozeile 1033
 - 18.1.2 Grafische Benutzeroberfläche 1033
 - 18.1.3 Abstract Window Toolkit (AWT) 1034
 - 18.1.4 Java Foundation Classes und Swing 1034
 - 18.1.5 JavaFX 1034
 - 18.1.6 SWT (Standard Widget Toolkit) * 1036
- 18.2 Deklarative und programmierte Oberflächen 1037
 - 18.2.1 GUI-Beschreibungen in JavaFX 1038
 - 18.2.2 Deklarative GUI-Beschreibungen für Swing? 1038
- 18.3 GUI-Builder 1038
 - 18.3.1 GUI-Builder für JavaFX 1039
 - 18.3.2 GUI-Builder für Swing 1039
- 18.4 Mit dem Eclipse WindowBuilder zur ersten Swing-Oberfläche 1039
 - 18.4.1 WindowBuilder installieren 1040
 - 18.4.2 Mit WindowBuilder eine GUI-Klasse hinzufügen 1041
 - 18.4.3 Das Layoutprogramm starten 1043

18.4.4	Grafische Oberfläche aufbauen	1044
18.4.5	Swing-Komponenten-Klassen	1047
18.4.6	Funktionalität geben	1048
18.5	Grundlegendes zum Zeichnen	1051
18.5.1	Die paint(Graphics)-Methode für das AWT-Frame	1051
18.5.2	Die ereignisorientierte Programmierung ändert Fensterinhalte	1053
18.5.3	Zeichnen von Inhalten auf einen JFrame	1054
18.5.4	Auffordern zum Neuzeichnen mit repaint(...)	1056
18.5.5	Java 2D-API	1056
18.6	Zum Weiterlesen	1057
19	Einführung in Dateien und Datenströme	1059
19.1	Alte und neue Welt in java.io und java.nio	1059
19.1.1	java.io-Paket mit File-Klasse	1059
19.1.2	NIO.2 und das java.nio-Paket	1060
19.1.3	java.io.File oder java.nio.*-Typen?	1060
19.2	Dateisysteme und Pfade	1061
19.2.1	FileSystem und Path	1061
19.2.2	Die Utility-Klasse Files	1067
19.3	Dateien mit wahlfreiem Zugriff	1070
19.3.1	Ein RandomAccessFile zum Lesen und Schreiben öffnen	1070
19.3.2	Aus dem RandomAccessFile lesen	1071
19.3.3	Schreiben mit RandomAccessFile	1074
19.3.4	Die Länge des RandomAccessFile	1074
19.3.5	Hin und her in der Datei	1075
19.4	Basisklassen für die Ein-/Ausgabe	1076
19.4.1	Die vier abstrakten Basisklassen	1076
19.4.2	Die abstrakte Basisklasse OutputStream	1077
19.4.3	Die abstrakte Basisklasse InputStream	1080
19.4.4	Die abstrakte Basisklasse Writer	1082
19.4.5	Die Schnittstelle Appendable *	1083
19.4.6	Die abstrakte Basisklasse Reader	1083
19.4.7	Die Schnittstellen Closeable, AutoCloseable und Flushable	1087
19.5	Lesen aus Dateien und Schreiben in Dateien	1088
19.5.1	Byteorientierte Datenströme über Files beziehen	1089
19.5.2	Zeichenorientierte Datenströme über Files beziehen	1089

19.5.3	Die Funktion von OpenOption bei den Files.newXXX(...)-Methoden	1091
19.5.4	Ressourcen aus dem Modulpfad und aus JAR-Dateien laden	1093
19.6	Zum Weiterlesen	1095
20	Einführung ins Datenbankmanagement mit JDBC	1097
20.1	Relationale Datenbanken und Java-Zugriffe	1097
20.1.1	Das relationale Modell	1097
20.1.2	Java-APIs zum Zugriff auf relationale Datenbanken	1098
20.1.3	Die JDBC-API und Implementierungen: JDBC-Treiber	1099
20.1.4	H2 ist das Werkzeug auf der Insel	1099
20.2	Eine Beispielabfrage	1100
20.2.1	Schritte zur Datenbankabfrage	1100
20.2.2	Mit Java auf die relationale Datenbank zugreifen	1100
20.3	Zum Weiterlesen	1102
21	Bits und Bytes, Mathematisches und Geld	1103
21.1	Bits und Bytes	1103
21.1.1	Die Bit-Operatoren Komplement, Und, Oder und XOR	1104
21.1.2	Repräsentation ganzer Zahlen in Java – das Zweierkomplement	1105
21.1.3	Das binäre (Basis 2), oktale (Basis 8), hexadezimale (Basis 16) Stellenwertsystem	1107
21.1.4	Auswirkung der Typumwandlung auf die Bit-Muster	1108
21.1.5	Vorzeichenlos arbeiten	1110
21.1.6	Die Verschiebeoperatoren	1113
21.1.7	Ein Bit setzen, löschen, umdrehen und testen	1116
21.1.8	Bit-Methoden der Integer- und Long-Klasse	1116
21.2	Fließkomma-Arithmetik in Java	1118
21.2.1	Spezialwerte für Unendlich, Null, NaN	1118
21.2.2	Standardnotation und wissenschaftliche Notation bei Fließkommazahlen *	1121
21.2.3	Mantisse und Exponent *	1122
21.3	Die Eigenschaften der Klasse Math	1124
21.3.1	Attribute	1124
21.3.2	Absolutwerte und Vorzeichen	1124

21.3.3	Maximum/Minimum	1125
21.3.4	Runden von Werten	1126
21.3.5	Rest der ganzzahligen Division *	1128
21.3.6	Division mit Rundung in Richtung negativ unendlich, alternativer Restwert *	1129
21.3.7	Multiply-Accumulate	1131
21.3.8	Wurzel- und Exponentialmethoden	1131
21.3.9	Der Logarithmus *	1132
21.3.10	Winkelmethode *	1133
21.3.11	Zufallszahlen	1134
21.4	Genauigkeit, Wertebereich eines Typs und Überlaufkontrolle *	1135
21.4.1	Der größte und der kleinste Wert	1135
21.4.2	Überlauf und alles ganz exakt	1136
21.4.3	Was bitte macht eine ulp?	1138
21.5	Zufallszahlen: Random, SecureRandom und SplittableRandom	1140
21.5.1	Die Klasse Random	1140
21.5.2	Random-Objekte mit dem Samen aufbauen	1140
21.5.3	Einzelne Zufallszahlen erzeugen	1141
21.5.4	Pseudo-Zufallszahlen in der Normalverteilung *	1142
21.5.5	Strom von Zufallszahlen generieren *	1142
21.5.6	Die Klasse SecureRandom *	1144
21.5.7	SplittableRandom *	1144
21.6	Große Zahlen *	1144
21.6.1	Die Klasse BigInteger	1145
21.6.2	Beispiel: Ganz lange Fakultäten mit BigInteger	1152
21.6.3	Große Fließkommazahlen mit BigDecimal	1153
21.6.4	Mit MathContext komfortabel die Rechengenauigkeit setzen	1156
21.6.5	Noch schneller rechnen durch mutable Implementierungen	1158
21.7	Mathe bitte strikt *	1158
21.7.1	Strikte Fließkommaberechnungen mit strictfp	1158
21.7.2	Die Klassen Math und StrictMath	1159
21.8	Geld und Währung	1159
21.8.1	Geldbeträge repräsentieren	1159
21.8.2	ISO 4217	1160
21.8.3	Währungen in Java repräsentieren	1161
21.9	Zum Weiterlesen	1161

22	Testen mit JUnit	1163
22.1	Softwaretests	1163
22.1.1	Vorgehen beim Schreiben von Testfällen	1164
22.2	Das Test-Framework JUnit	1164
22.2.1	Test-Driven Development und Test-First	1165
22.2.2	Testen, implementieren, testen, implementieren, testen, freuen	1166
22.2.3	JUnit-Tests ausführen	1168
22.2.4	assertXXX(..)-Methoden der Klasse Assertions	1169
22.2.5	Exceptions testen	1171
22.2.6	Grenzen für Ausführungszeiten festlegen	1173
22.2.7	Beschriftungen mit @DisplayName	1173
22.2.8	Verschachtelte Tests	1174
22.2.9	Tests ignorieren	1174
22.2.10	Mit Methoden der Assumptions-Klasse Tests abbrechen	1175
22.2.11	Parametrisierte Tests	1175
22.3	Java-Assertions-Bibliotheken und AssertJ	1176
22.3.1	AssertJ	1177
22.4	Aufbau größerer Testfälle	1178
22.4.1	Fixtures	1179
22.4.2	Sammlungen von Testklassen und Klassenorganisation	1181
22.5	Wie gutes Design das Testen ermöglicht	1181
22.6	Dummy, Fake, Stub und Mock	1183
22.7	JUnit-Erweiterungen, Testzusätze	1185
22.8	Zum Weiterlesen	1186
23	Die Werkzeuge des JDK	1187
23.1	Übersicht	1187
23.1.1	Aufbau und gemeinsame Schalter	1188
23.2	Java-Quellen übersetzen	1188
23.2.1	Der Java-Compiler des JDK	1188
23.2.2	Alternative Compiler	1189
23.2.3	Native Compiler	1190
23.3	Die Java-Laufzeitumgebung	1190
23.3.1	Schalter der JVM	1191
23.3.2	Der Unterschied zwischen java.exe und javaw.exe	1193

23.4 Dokumentationskommentare mit Javadoc 1194

23.4.1 Einen Dokumentationskommentar setzen 1194

23.4.2 Mit dem Werkzeug javadoc eine Dokumentation erstellen 1196

23.4.3 HTML-Tags in Dokumentationskommentaren * 1197

23.4.4 Generierte Dateien 1197

23.4.5 Dokumentationskommentare im Überblick * 1198

23.4.6 Javadoc und Doclets * 1200

23.4.7 Veraltete (deprecated) Typen und Eigenschaften 1200

23.4.8 Javadoc-Überprüfung mit DocLint 1203

23.5 Das Archivformat JAR 1204

23.5.1 Das Dienstprogramm jar benutzen 1204

23.5.2 Das Manifest 1207

23.5.3 Applikationen in JAR-Archiven starten 1207

23.6 jlink: der Java Linker 1209

23.7 Zum Weiterlesen 1210

Anhang 1211

Index 1229