

Java ist auch eine Insel

Einführung, Ausbildung, Praxis

DAS INHALTS- VERZEICHNIS

» Hier geht's
direkt
zum Buch

Auf einen Blick

1	Java ist auch eine Sprache	49
2	Imperative Sprachkonzepte	91
3	Klassen und Objekte	221
4	Arrays und ihre Anwendungen	265
5	Der Umgang mit Zeichen und Zeichenketten	311
6	Eigene Klassen schreiben	395
7	Objektorientierte Beziehungsfragen	459
8	Records, Schnittstellen, Aufzählungen, versiegelte Klassen	523
9	Ausnahmen müssen sein	597
10	Geschachtelte Typen	661
11	Besondere Typen der Java SE	681
12	Generics<T>	751
13	Lambda-Ausdrücke und funktionale Programmierung	807
14	Architektur, Design und angewandte Objektorientierung	877
15	Java Platform Module System	891
16	Die Klassenbibliothek	913
17	Einführung in die nebenläufige Programmierung	947
18	Einführung in Datenstrukturen und Algorithmen	983
19	Einführung in grafische Oberflächen	1039
20	Einführung in Dateien und Datenströme	1049
21	Einführung ins Datenbankmanagement mit JDBC	1087
22	Bits und Bytes, Mathematisches und Geld	1093
23	Testen mit JUnit	1153
24	Die Werkzeuge des JDK	1179

Inhalt

Materialien zum Buch	32
Vorwort	33

1 Java ist auch eine Sprache 49

1.1 Historischer Hintergrund	50
1.1.1 Wo ist die Sonne? Oracle übernimmt Sun Microsystems 2010	51
1.2 Warum Java populär ist – die zentralen Eigenschaften	52
1.2.1 Bytecode	52
1.2.2 Ausführung des Bytecodes durch eine virtuelle Maschine	53
1.2.3 Plattformunabhängigkeit	53
1.2.4 Java als Sprache, Laufzeitumgebung und Standardbibliothek	53
1.2.5 Objektorientierung in Java	54
1.2.6 Java ist verbreitet und bekannt	54
1.2.7 Java ist schnell – Optimierung und Just-in-time-Compilation	55
1.2.8 Zeiger und Referenzen	57
1.2.9 Bring den Müll raus, Garbage-Collector!	58
1.2.10 Ausnahmebehandlung	59
1.2.11 Das Angebot an Bibliotheken und Werkzeugen	60
1.2.12 Vergleichbar einfache Syntax	60
1.2.13 Verzicht auf umstrittene Konzepte	61
1.2.14 Java ist Open Source	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 Java und C#/.NET	64
1.4 Einschränkungen von Java	65
1.4.1 Wofür sich Java weniger eignet	65
1.4.2 Alternative Sprachen und Technologien	66
1.4.3 Brücken zur Systemnähe in Java	66
1.5 Weiterentwicklung und Umbrüche	67
1.5.1 Die Entwicklung von Java und seine Zukunftsaussichten	67
1.5.2 Features, Enhancements (Erweiterungen) und ein JSR	68
1.5.3 Applets	69
1.5.4 JavaFX	69

1.6 Java-Plattformvarianten	70
1.6.1 Die Java SE-Plattform	70
1.6.2 Java ME: Java für die Kleinen	73
1.6.3 Java für die ganz, ganz Kleinen	74
1.6.4 Jakarta EE: Java für die Großen	74
1.7 Java SE-Implementierungen	75
1.7.1 OpenJDK	75
1.7.2 Oracle JDK	76
1.8 Installation des JDK und erster Start	78
1.8.1 Oracle JDK unter Windows installieren	78
1.8.2 Das erste Programm compilieren und testen	80
1.8.3 Der Compilerlauf	81
1.8.4 Die Laufzeitumgebung	82
1.8.5 Häufige Compiler- und Interpreter-Probleme	82
1.9 Entwicklungsumgebungen	84
1.9.1 IntelliJ IDEA	84
1.9.2 Eclipse IDE	88
1.9.3 NetBeans	90
1.10 Zum Weiterlesen	90

2 Imperative Sprachkonzepte

2.1 Von der Klasse zur Anweisung	91
2.1.1 Was sind Anweisungen?	92
2.1.2 Die Reise beginnt am main	92
2.1.3 Der erste Methodenaufruf: println(..)	94
2.1.4 (Reservierte) Schlüsselwörter	95
2.1.5 Bezeichner	97
2.1.6 Literale	99
2.1.7 Atomare Anweisungen und Anweisungssequenzen	100
2.1.8 Mehr zu print(..), println(..) und printf(..) für Bildschirmausgaben	100
2.1.9 Die API-Dokumentation	103
2.1.10 Ausdrücke	105
2.1.11 Ausdrucksanweisung	105
2.1.12 Erster Einblick in die Objektorientierung	106
2.1.13 Modifizierer	107
2.1.14 Gruppieren von Anweisungen mit Blöcken	107

2.2 Lexikalische Grundlagen	109
2.2.1 Tokens	109
2.2.2 Textkodierung durch Unicode-Zeichen	110
2.2.3 Zusammenfassung der lexikalischen Analyse	111
2.2.4 Kommentare	111
2.3 Datentypen, Typisierung, Variablen und Zuweisungen	113
2.3.1 Primitiv- oder Verweistyp	114
2.3.2 Primitive Datentypen im Überblick	115
2.3.3 Variablen Deklarationen	118
2.3.4 Automatisches Feststellen der Typen mit var	120
2.3.5 Finale Variablen und der Modifizierer final	121
2.3.6 Unbenannte Variablen mit _	122
2.3.7 Konsoleneingaben	123
2.3.8 Wahrheitswerte	125
2.3.9 Ganzzahlige Datentypen	125
2.3.10 Unterstriche in Zahlen	127
2.3.11 Alphanumerische Zeichen	128
2.3.12 Gleitkommazahlen mit den Datentypen float und double	129
2.3.13 Gute Namen, schlechte Namen	130
2.3.14 Keine automatische Initialisierung von lokalen Variablen	131
2.4 Ausdrücke, Operanden und Operatoren	132
2.4.1 Die Arten von Operatoren	132
2.4.2 Zuweisungsoperator	133
2.4.3 Arithmetische Operatoren	134
2.4.4 Unäres Minus und Plus	138
2.4.5 Präfix- oder Postfix-Inkrement und -Dekrement	139
2.4.6 Zuweisung mit Operation (Verbundoperator)	141
2.4.7 Die relationalen Operatoren und die Gleichheitssoperatoren	142
2.4.8 Logische Operatoren: Nicht, Und, Oder, XOR	143
2.4.9 Kurzschluss-Operatoren	144
2.4.10 Der Rang der Operatoren in der Auswertungsreihenfolge	146
2.4.11 Die Typumwandlung (das Casting)	149
2.4.12 Überladenes Plus für Strings	154
2.4.13 Operator vermisst *	155
2.5 Bedingte Anweisungen oder Fallunterscheidungen	156
2.5.1 Verzweigung mit der if-Anweisung	156
2.5.2 Die Alternative mit einer if-else-Anweisung wählen	159
2.5.3 Der Bedingungsoperator	162
2.5.4 Die switch-Anweisung bietet die Alternative	165
2.5.5 Switch-Ausdrücke	172

2.6	Immer das Gleiche mit den Schleifen	175
2.6.1	Die while-Schleife	176
2.6.2	Die do-while-Schleife	177
2.6.3	Die for-Schleife	179
2.6.4	Schleifenbedingungen und Vergleiche mit == *	184
2.6.5	Schleifenabbruch mit break und zurück zum Test mit continue	186
2.6.6	break und continue mit Marken *	189
2.7	Methoden deklarieren	193
2.7.1	Bestandteile einer Methode	193
2.7.2	Methoden ohne Parameter deklarieren	195
2.7.3	Parameter, Argument und Wertübergabe	196
2.7.4	Signatur-Beschreibung in der Java-API	199
2.7.5	Methoden vorzeitig mit return beenden	201
2.7.6	Methoden mit Rückgaben	201
2.7.7	Methoden überladen	203
2.7.8	Vorgegebener Wert für nicht aufgeführte Argumente	205
2.7.9	Rückgabepfade und Kontrollfluss in Methoden	206
2.7.10	Rekursive Methoden *	210
2.8	Methoden brauchen ein Zuhause: Instanz oder Klasse	214
2.8.1	Statische Methoden (Klassenmethoden)	214
2.8.2	Kompakte Quelldatei und »echte« Klassen	215
2.8.3	Statische Methoden von anderen Klassen aufrufen	217
2.8.4	Gültigkeitsbereich	218
2.9	Zum Weiterlesen	220

3 Klassen und Objekte

3.1	Objektorientierte Programmierung (OOP)	221
3.1.1	Warum überhaupt OOP?	222
3.1.2	Denk ich an Java, denk ich an Wiederverwendbarkeit	222
3.2	Eigenschaften einer Klasse	223
3.2.1	Klassenarbeit mit Point	224
3.3	Natürlich modellieren mit der UML (Unified Modeling Language) *	225
3.3.1	Wichtige Diagrammtypen der UML *	225
3.4	Neue Objekte erzeugen	227
3.4.1	Ein Objekt einer Klasse mit dem Schlüsselwort new anlegen	227
3.4.2	Deklarieren von Referenzvariablen	227

3.4.3	Jetzt mach mal 'nen Punkt: Zugriff auf Objektvariablen und -methoden	228
3.4.4	Überblick über Point-Methoden	232
3.4.5	Konstruktoren nutzen	236
3.5	ZZZZZnake	237
3.5.1	Erweiterung	239
3.6	Pakete schnüren, Importe und Compilationseinheiten	239
3.6.1	Java-Pakete	239
3.6.2	Pakete der Standardbibliothek	240
3.6.3	Volle Qualifizierung und import-Deklaration	240
3.6.4	Mit import p1.p2.* alle Typen eines Pakets erreichen	242
3.6.5	Modul-Import	243
3.6.6	Hierarchische Strukturen über Pakete und die Spiegelung im Dateisystem	246
3.6.7	Die package-Deklaration	246
3.6.8	Unbenanntes Paket (default package)	248
3.6.9	Compilationseinheit (Compilation Unit)	248
3.6.10	Statischer Import *	249
3.7	Mit Referenzen arbeiten, Vielfalt, Identität, Gleichwertigkeit	250
3.7.1	null-Referenz und die Frage der Philosophie	250
3.7.2	Alles auf null? Referenzen testen	252
3.7.3	Zuweisungen bei Referenzen	253
3.7.4	Methoden mit Referenztypen als Parameter	255
3.7.5	Identität von Objekten	259
3.7.6	Gleichwertigkeit und die Methode equals(...)	259
3.7.7	Ausblick: Value Types – Objekte ohne Identität	261
3.8	Der Zusammenhang von new, Heap und Garbage-Collector	262
3.8.1	Heap-Speicher	262
3.8.2	Automatische Speicherbereinigung/Garbage-Collector (GC) – es ist dann mal weg	263
3.8.3	OutOfMemoryError	263
3.9	Zum Weiterlesen	264

4 Arrays und ihre Anwendungen

4.1	Einfache Feldarbeit	265
4.1.1	Grundbestandteile	266
4.1.2	Deklaration von Array-Variablen	267
4.1.3	Array-Objekte mit new erzeugen	268

4.1.4	Arrays mit { Inhalt }	269
4.1.5	Die Länge eines Arrays über die Objektvariable length auslesen	270
4.1.6	Zugriff auf die Elemente über den Index	270
4.1.7	Typische Array-Fehler	272
4.1.8	Arrays an Methoden übergeben	274
4.1.9	Mehrere Rückgabewerte *	274
4.1.10	Vorinitialisierte Arrays	275
4.2	Die erweiterte for-Schleife	277
4.2.1	Anonyme Arrays in der erweiterten for-Schleife nutzen	278
4.2.2	Umsetzung und Einschränkung	278
4.2.3	Beispiel: Arrays mit Strings durchsuchen	279
4.2.4	Zufällige Spielerpositionen erzeugen	279
4.3	Methode mit variabler Argumentanzahl (Varargs)	281
4.3.1	System.out.printf(...) nimmt eine beliebige Anzahl von Argumenten an	281
4.3.2	Durchschnitt finden von variablen Argumenten	282
4.3.3	Varargs-Designtipps	283
4.4	Mehrdimensionale Arrays *	284
4.4.1	Mehrdimensionale Array-Objekte mit new aufbauen	284
4.4.2	Anlegen und Initialisieren in einem Schritt	284
4.4.3	Zugriff auf Elemente	285
4.4.4	length bei mehrdimensionalen Arrays	285
4.4.5	Zweidimensionale Arrays mit ineinander verschachtelten Schleifen ablaufen	285
4.4.6	Nichtrechteckige Arrays *	287
4.5	Bibliotheksunterstützung von Arrays	290
4.5.1	Klonen kann sich lohnen – Arrays vermehren	290
4.5.2	Warum »können« Arrays so wenig?	291
4.5.3	Array-Inhalte kopieren	291
4.6	Die Klasse Arrays zum Vergleichen, Füllen, Suchen und Sortieren nutzen	292
4.6.1	String-Präsentation eines Arrays	293
4.6.2	Sortieren	293
4.6.3	Arrays von Primitiven mit Arrays.equals(...) und Arrays.deepEquals(...) vergleichen *	295
4.6.4	Objekt-Arrays mit Arrays.equals(...) und Arrays.deepEquals(...) vergleichen *	296
4.6.5	Unterschiede suchen mit mismatch (...) *	297
4.6.6	Füllen von Arrays *	298
4.6.7	Array-Abschnitte kopieren *	299
4.6.8	Halbierungssuche *	300

4.6.9	Array-Vergleiche mit compare(...) und compareUnsigned(...)	302
4.6.10	Arrays zu Listen mit Arrays.asList(...) – praktisch für die Suche und zum Vergleichen *	302
4.6.11	Eine lange Schlange	304
4.7	Der Startpunkt für das Laufzeitsystem: main(...)	307
4.7.1	Kommandozeilenargumente	307
4.7.2	Deklaration der Startmethode mit Array für Argumente	307
4.7.3	Kommandozeilenargumente verarbeiten	308
4.7.4	Der Rückgabetyp von main(...) und System.exit(int) *	309
4.8	Zum Weiterlesen	310

5 Der Umgang mit Zeichen und Zeichenketten 311

5.1	Von ASCII über ISO-8859-1 zu Unicode	311
5.1.1	ASCII	311
5.1.2	ISO/IEC 8859-1	312
5.1.3	Unicode	313
5.1.4	Unicode-Zeichenkodierung	314
5.1.5	Escape-Sequenzen/Fluchtsymbole	315
5.1.6	Schreibweise für Unicode-Zeichen und Unicode-Escapes	316
5.1.7	Java-Versionen gehen mit dem Unicode-Standard Hand in Hand *	318
5.2	Datentypen für Zeichen und Zeichenketten	319
5.3	Die Character-Klasse	320
5.3.1	Ist das so?	320
5.3.2	Zeichen in Großbuchstaben/Kleinbuchstaben konvertieren	322
5.3.3	Vom Zeichen zum String	323
5.3.4	Von char in int: vom Zeichen zur Zahl *	323
5.4	Datentypen für Zeichenketten	325
5.4.1	Die Klasse String	326
5.4.2	Die Klassen StringBuilder/StringBuffer	326
5.4.3	Der Basistyp CharSequence für Zeichenketten	327
5.4.4	Enthält ein String ein char-Array? *	327
5.5	Die Klasse String und ihre Methoden	327
5.5.1	String-Literale als String-Objekte für konstante Zeichenketten	328
5.5.2	Konkatenation mit +	328
5.5.3	Mehrzeilige Textblöcke mit """	328

5.5.4	String-Länge und Test auf Leer-String	333
5.5.5	Zugriff auf ein bestimmtes Zeichen mit charAt(int)	335
5.5.6	Nach enthaltenen Zeichen und Zeichenketten suchen	336
5.5.7	Das Hangman-Spiel	339
5.5.8	Gut, dass wir verglichen haben	341
5.5.9	String-Teile extrahieren	345
5.5.10	Strings anhängen, zusammenfügen, Groß-/Kleinschreibung und Weißraum	350
5.5.11	Gesucht, gefunden, ersetzt	353
5.5.12	String-Objekte mit Konstruktoren und aus Wiederholungen erzeugen *	355
5.6	Veränderbare Zeichenketten mit StringBuilder	359
5.6.1	Überblick	359
5.6.2	Anlegen von StringBuilder-Objekten	359
5.6.3	StringBuilder in andere Zeichenkettenformate konvertieren	360
5.6.4	Zeichen(folgen) erfragen	360
5.6.5	Daten anhängen	360
5.6.6	Zeichen(folgen) setzen, löschen und umdrehen	362
5.6.7	Länge und Kapazität eines StringBuilder-Objekts *	364
5.6.8	Vergleich von StringBuilder-Instanzen und Strings mit StringBuilder	365
5.6.9	hashCode() bei StringBuilder *	367
5.7	CharSequence als Basistyp	368
5.7.1	Basisoperationen der Schnittstelle	369
5.7.2	Statische compare(..)-Methode in CharSequence	370
5.7.3	Default-Methoden in der Schnittstelle CharSequence *	370
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 in den Formaten Binär, Hex und Oktal *	375
5.9	Strings verketten (konkatenieren)	379
5.9.1	Strings mit StringJoiner verketten	379
5.10	Zerlegen von Zeichenketten	381
5.10.1	Splitten von Zeichenketten mit split(..)	382
5.10.2	Yes we can, yes we scan – die Klasse Scanner	383
5.11	Ausgaben formatieren	386
5.11.1	Formatieren und Ausgeben mit format(..)	386
5.12	Zum Weiterlesen	393

6 Eigene Klassen schreiben

395

6.1	Eigene Klassen mit Eigenschaften deklarieren	395
6.1.1	Minimalklasse	396
6.1.2	Objektvariablen deklarieren	396
6.1.3	Methoden deklarieren	399
6.1.4	Verdeckte (shadowed) Variablen	402
6.1.5	Die this-Referenz	403
6.2	Privatsphäre und Sichtbarkeit	407
6.2.1	Für die Öffentlichkeit: public	408
6.2.2	Kein Public Viewing – Passwörter sind privat	408
6.2.3	Wieso nicht freie Methoden und Variablen für alle?	410
6.2.4	Privat ist nicht ganz privat: Es kommt darauf an, wer's sieht *	410
6.2.5	Zugriffsmethoden für Objektvariablen deklarieren	411
6.2.6	Setter und Getter nach der JavaBeans-Spezifikation	412
6.2.7	Paketsichtbar	414
6.2.8	Zusammenfassung zur Sichtbarkeit	415
6.3	Eine für alle – statische Methoden und Klassenvariablen	418
6.3.1	Warum statische Eigenschaften sinnvoll sind	419
6.3.2	Statische Eigenschaften mit static	419
6.3.3	Statische Eigenschaften über Referenzen nutzen? *	422
6.3.4	Warum die Groß- und Kleinschreibung wichtig ist *	423
6.3.5	Statische Variablen zum Datenaustausch *	424
6.3.6	Statische Eigenschaften und Objekteigenschaften *	425
6.4	Konstanten und Aufzählungen	426
6.4.1	Konstanten über statische finale Variablen	426
6.4.2	Typsichere Aufzählungen	428
6.4.3	Aufzählungstypen: typsichere Aufzählungen mit enum	430
6.5	Objekte anlegen und zerstören	435
6.5.1	Konstruktoren schreiben	435
6.5.2	Verwandtschaft von Methode und Konstruktor	437
6.5.3	Der Standard-Konstruktor (default constructor)	437
6.5.4	Parametrisierte und überladene Konuktoren	438
6.5.5	Copy-Konstruktor	441
6.5.6	Einen anderen Konstruktor der gleichen Klasse mit this(..) aufrufen	442
6.5.7	Immutable-Objekte und Wither-Methoden	445
6.5.8	Ihr fehlt uns nicht – der Garbage-Collector	447

6.6	Klassen- und Objektinitialisierung *	449
6.6.1	Initialisierung von Objektvariablen	449
6.6.2	Statische Blöcke als Klasseninitialisierer	451
6.6.3	Initialisierung von Klassenvariablen	452
6.6.4	Eincompilierte Belegungen der finalen Klassenvariablen	453
6.6.5	Exemplarinitialisierer (Instanzinitialisierer)	454
6.6.6	Finale Werte im Konstruktor und in statischen Blöcken setzen	457
6.7	Zum Weiterlesen	458
7	Objektorientierte Beziehungsfragen	459
7.1	Assoziationen zwischen Objekten	459
7.1.1	Assoziationsarten	459
7.1.2	Unidirektionale 1:1-Beziehung	460
7.1.3	Zwei Freunde müsst ihr werden – bidirektionale 1:1-Beziehungen	461
7.1.4	Unidirektionale 1:n-Beziehung	463
7.2	Vererbung	470
7.2.1	Vererbung in Java	471
7.2.2	Ereignisse modellieren	471
7.2.3	Die implizite Basisklasse <code>java.lang.Object</code>	473
7.2.4	Einfach- und Mehrfachvererbung *	474
7.2.5	Sehen Kinder alles? Die Sichtbarkeit <code>protected</code>	475
7.2.6	Konstruktoren in der Vererbung und <code>super(...)</code>	476
7.3	Typen in Hierarchien	481
7.3.1	Automatische und explizite Typumwandlung	481
7.3.2	Das Substitutionsprinzip	484
7.3.3	Typen mit dem <code>instanceof</code> -Operator testen	487
7.3.4	Pattern-Matching bei <code>instanceof</code>	489
7.3.5	Pattern-Matching bei <code>switch</code>	491
7.4	Methoden überschreiben	494
7.4.1	Methoden in Unterklassen mit neuem Verhalten ausstatten	494
7.4.2	Mit <code>super</code> an die Eltern	498
7.5	Drum prüfe, wer sich dynamisch bindet	501
7.5.1	Gebunden an <code>toString()</code>	501
7.5.2	Implementierung von <code>System.out.println(Object)</code>	503

7.6 Finale Klassen und finale Methoden	504
7.6.1 Finale Klassen	504
7.6.2 Nicht überschreibbare (finale) Methoden	504
7.7 Abstrakte Klassen und abstrakte Methoden	506
7.7.1 Abstrakte Klassen	506
7.7.2 Abstrakte Methoden	508
7.8 Weiteres zum Überschreiben und dynamischen Binden	514
7.8.1 Nicht dynamisch gebunden bei privaten, statischen und finalen Methoden	515
7.8.2 Kovariante Rückgabetypen	515
7.8.3 Array-Typen und Kovarianz *	516
7.8.4 Dynamisch gebunden auch bei Konstruktoraufrufen *	517
7.8.5 Keine dynamische Bindung bei überdeckten Objektvariablen *	519
7.9 Zum Weiterlesen und Programmieraufgabe	520

8 Records, Schnittstellen, Aufzählungen, versiegelte Klassen

523

8.1 Records	523
8.1.1 Einfache Records	523
8.1.2 Records mit Methoden	525
8.1.3 Konuktoren von Records anpassen	527
8.1.4 Konuktoren ergänzen	529
8.1.5 Record-Patterns zur Destrukturierung	530
8.1.6 Records: Zusammenfassung	534
8.2 Schnittstellen	535
8.2.1 Schnittstellen sind neue Typen	535
8.2.2 Schnittstellen deklarieren	535
8.2.3 Abstrakte Methoden in Schnittstellen	536
8.2.4 Schnittstellen implementieren	537
8.2.5 Ein Polymorphie-Beispiel mit Schnittstellen	540
8.2.6 Klassen können eine Oberklasse haben und Schnittstellen implementieren	543
8.2.7 Records implementieren Schnittstellen	546
8.2.8 Die Mehrfachvererbung bei Schnittstellen	547
8.2.9 Erweitern von Interfaces – Unterschnittstellen	550
8.2.10 Geldkraken im Direktvergleich – mit der Schnittstelle Comparable	554

8.2.11	Konstantendeklarationen bei Schnittstellen	556
8.2.12	Nachträgliches Implementieren von Schnittstellen *	557
8.2.13	Statische ausprogrammierte Methoden in Schnittstellen	558
8.2.14	Erweitern und Ändern von Schnittstellen	559
8.2.15	Default-Methoden	561
8.2.16	Öffentliche und private Schnittstellenmethoden	567
8.2.17	Erweiterte Schnittstellen, Mehrfachvererbung und Mehrdeutigkeiten *	567
8.2.18	Bausteine bilden mit Default-Methoden *	572
8.2.19	Markierungsschnittstellen *	575
8.2.20	(Abstrakte) Klassen und Schnittstellen im Vergleich	575
8.3	Aufzählungstypen	576
8.3.1	Existierende Methoden auf Enum-Objekten	577
8.3.2	Aufzählungen mit eigenen Methoden *	580
8.3.3	enum mit eigenen Konstruktoren und Initialisierern	583
8.3.4	enum kann Schnittstellen implementieren	586
8.4	Versiegelte Klassen und Schnittstellen	587
8.4.1	Grenzen von Aufzählungstypen und regulären Klassen	588
8.4.2	Versiegelte Klassen und Schnittstellen (sealed classes/interfaces)	589
8.4.3	Unterklassen sind final, sealed, non-sealed	591
8.4.4	Vollständige Abdeckung von Aufzählungen	592
8.4.5	Abkürzende Schreibweisen	593
8.4.6	Versiegelte Schnittstellen und Records	594
8.5	Zum Weiterlesen	595

9	Ausnahmen müssen sein	597
9.1	Problembereiche einzäunen	598
9.1.1	Exceptions in Java mit try und catch	598
9.1.2	Geprüfte und ungeprüfte Ausnahmen	599
9.1.3	Eine NumberFormatException fliegt (ungeprüfte Ausnahme)	599
9.1.4	Datum-Zeitstempel in Textdatei anhängen (geprüfte Ausnahme)	601
9.1.5	Wiederholung abgebrochener Bereiche *	604
9.1.6	Bitte nicht schlucken – leere catch-Blöcke	605
9.1.7	Mehrere Ausnahmen auffangen	605
9.1.8	Zusammenfassen gleicher catch-Blöcke mit dem multi-catch	606
9.2	Ausnahmen mit throws nach oben reichen	607
9.2.1	throws bei Konstruktoren und Methoden	607

9.3 Die Klassenhierarchie der Ausnahmen	608
9.3.1 Eigenschaften des Ausnahmeobjekts	608
9.3.2 Basistyp Throwable	610
9.3.3 Die Exception-Hierarchie	610
9.3.4 Oberausnahmen auffangen	611
9.3.5 Schon gefangen?	613
9.3.6 Ablauf einer Ausnahmesituation	614
9.3.7 Nicht zu allgemein fangen!	614
9.3.8 Bekannte RuntimeException-Klassen	616
9.3.9 Kann man abfangen, muss man aber nicht	617
9.4 Abschlussbehandlung mit finally	618
9.4.1 Die ignorante Version	618
9.4.2 Der gut gemeinte Versuch	619
9.4.3 Ab jetzt wird scharf geschlossen	620
9.4.4 Zusammenfassung	621
9.4.5 Ein try ohne catch, aber ein try-finally	622
9.5 Auslösen eigener Exceptions	623
9.5.1 Mit throw Ausnahmen auslösen	623
9.5.2 Vorhandene ungeprüfte Ausnahmetypen kennen und nutzen	625
9.5.3 Parameter testen und gute Fehlermeldungen	628
9.5.4 Neue Exception-Klassen deklarieren	630
9.5.5 Eigene Ausnahmen als Unterklassen von Exception oder RuntimeException?	631
9.5.6 Ausnahmen abfangen und weiterleiten *	634
9.5.7 Aufruf-Stack von Ausnahmen verändern *	635
9.5.8 Geschachtelte Ausnahmen *	636
9.6 try mit Ressourcen (automatisches Ressourcenmanagement)	640
9.6.1 Ressourcenmanagement mit try mit Ressourcen	640
9.6.2 Die Schnittstelle AutoCloseable	641
9.6.3 Ausnahmen vom close()	642
9.6.4 Typen, die AutoCloseable und Closeable sind	643
9.6.5 Mehrere Ressourcen nutzen	645
9.6.6 try mit Ressourcen auf null-Ressourcen	646
9.6.7 Unterdrückte Ausnahmen *	646
9.7 Besonderheiten bei der Ausnahmebehandlung *	650
9.7.1 Rückgabewerte bei ausgelösten Ausnahmen	650
9.7.2 Ausnahmen und Rückgaben verschwinden – das Duo return und finally	650
9.7.3 throws bei überschriebenen Methoden	652
9.7.4 Nicht erreichbare catch-Klauseln	654

9.8	Harte Fehler – Error *	655
9.9	Zusicherungen im Programmcode (Assertions) *	656
9.9.1	Der assert-Anweisung	656
9.9.2	Assertions aktivieren	657
9.9.3	Feinsteuerung von Assertions	658
9.10	Zum Weiterlesen	659

10 Geschachtelte Typen

10.1	Geschachtelte Klassen, Records, Schnittstellen und Aufzählungen	661
10.2	Statische geschachtelte Typen	663
10.2.1	Modifizierer und Sichtbarkeit	664
10.2.2	Records als Behälter	664
10.2.3	Umsetzung von statischen geschachtelten Typen *	664
10.3	Nichtstatische geschachtelte Typen	665
10.3.1	Eine Schokoladenfabrik mit Schokoladenpresse	665
10.3.2	Exemplare innerer Klassen erzeugen	666
10.3.3	Die this-Referenz	666
10.3.4	Vom Compiler generierte Klassendateien *	668
10.4	Lokale Klassen	668
10.4.1	Beispiel mit eigener Klassendeklaration	668
10.4.2	Lokale Klasse für einen Timer nutzen	669
10.5	Anonyme innere Klassen	670
10.5.1	Nutzung einer anonymen inneren Klasse für den Timer	671
10.5.2	Umsetzung innerer anonymer Klassen *	672
10.5.3	Konstruktoren innerer anonymer Klassen	672
10.6	Vermischtes	675
10.6.1	Zugriff auf lokale Variablen aus lokalen und anonymen Klassen *	675
10.6.2	this in Unterklassen *	675
10.6.3	Geschachtelte Klassen greifen auf private Eigenschaften zu	676
10.6.4	Nester	678
10.7	Zum Weiterlesen	679

11 Besondere Typen der Java SE

681

11.1 Object ist die Mutter aller Klassen	682
11.1.1 Klassenobjekte	682
11.1.2 Objektidentifikation mit <code>toString()</code>	683
11.1.3 Objektgleichwertigkeit mit <code>equals(...)</code> und Identität	685
11.1.4 Klonen eines Objekts mit <code>clone()</code> *	691
11.1.5 Hashwerte über <code>hashCode()</code> liefern *	696
11.1.6 System.identityHashCode(...) und das Problem der nicht eindeutigen Objektverweise *	702
11.1.7 Synchronisation *	703
11.2 Schwache Referenzen und Cleaner	704
11.3 Die Utility-Klasse <code>java.util.Objects</code>	705
11.3.1 Eingebaute null-Tests für <code>equals(...)/hashCode()</code>	705
11.3.2 <code>Objects.toString(..)</code>	706
11.3.3 null-Prüfungen mit eingebauter Ausnahmebehandlung	707
11.3.4 Tests auf null	708
11.3.5 Indexbezogene Programmargumente auf Korrektheit prüfen	708
11.4 Vergleichen von Objekten und Ordnung herstellen	709
11.4.1 Natürlich geordnet oder nicht?	709
11.4.2 <code>compare*()</code> -Methode der Schnittstellen Comparable und Comparator	711
11.4.3 Rückgabewerte kodieren die Ordnung	711
11.4.4 Mit einem Beispiel-Comparator Süßigkeiten nach Kalorien sortieren	712
11.4.5 Tipps für Comparator- und Comparable-Implementierungen	714
11.4.6 Statische und Default-Methoden in Comparator	715
11.5 Wrapper-Klassen und Autoboxing	718
11.5.1 Wrapper-Objekte erzeugen	720
11.5.2 Konvertierungen in eine String-Repräsentation	721
11.5.3 Von einer String-Repräsentation parsen	722
11.5.4 Die Basisklasse Number für numerische Wrapper-Objekte	723
11.5.5 Vergleiche durchführen mit <code>compare*(...), compareTo(...), equals(...)</code> und Hashwerten	724
11.5.6 Statische Reduzierungsmethoden in Wrapper-Klassen	727
11.5.7 Konstanten für die Größe eines primitiven Typs	728
11.5.8 Behandeln von vorzeichenlosen Zahlen *	729
11.5.9 Die Klassen Integer und Long	730
11.5.10 Die Klassen Double und Float für Gleitkommazahlen	732
11.5.11 Die Boolean-Klasse	732
11.5.12 Autoboxing: Boxing und Unboxing	733

11.6 Iterator, Iterable *	737
11.6.1 Die Schnittstelle Iterator	738
11.6.2 Wer den Iterator liefert	741
11.6.3 Die Schnittstelle Iterable	741
11.6.4 Erweitertes for und Iterable	742
11.6.5 Interne Iteration	742
11.6.6 Ein eigenes Iterable implementieren *	743
11.7 Annotationen in der Java SE	745
11.7.1 Orte für Annotationen	745
11.7.2 Annotationstypen aus java.lang	746
11.7.3 @Deprecated	747
11.7.4 Annotationen mit zusätzlichen Informationen	747
11.7.5 @SuppressWarnings	748
11.8 Zum Weiterlesen	750

12 Generics<T>

12.1 Einführung in Java Generics	751
12.1.1 Mensch versus Maschine – Typprüfung des Compilers und der Laufzeitumgebung	751
12.1.2 Raketen	752
12.1.3 Generische Typen deklarieren	754
12.1.4 Generics nutzen	756
12.1.5 Diamonds are forever	759
12.1.6 Generische Schnittstellen	762
12.1.7 Generische Methoden/Konstruktoren und Typ-Inferenz	764
12.2 Umsetzen der Generics, Typlösung und Raw-Types	768
12.2.1 Realisierungsmöglichkeiten	768
12.2.2 Typlösung (Type Erasure)	768
12.2.3 Raw-Type	769
12.2.4 Probleme der Typlösung	772
12.3 Die Typen über Bounds einschränken	776
12.3.1 Einfache Einschränkungen mit extends	777
12.3.2 Weitere Obertypen mit &	780
12.4 Typparameter in der throws-Klausel *	780
12.4.1 Deklaration einer Klasse mit Typvariable <E extends Exception>	780
12.4.2 Parametrisierter Typ bei Typvariable <E extends Exception>	781

12.5 Generics und Vererbung, Invarianz	784
12.5.1 Arrays sind kovariant	784
12.5.2 Generics sind nicht kovariant, sondern invariant	784
12.5.3 Wildcards mit ?	785
12.5.4 Bounded Wildcards	788
12.5.5 Bounded-Wildcard-Typen und Bounded-Typvariablen	791
12.5.6 Das LESS-Prinzip	794
12.6 Konsequenzen der Typlösung: Typ-Token, Arrays und Brücken *	796
12.6.1 Typ-Token	796
12.6.2 Super-Type-Token	798
12.6.3 Generics und Arrays	799
12.6.4 Brückenmethoden *	800
12.7 Zum Weiterlesen	806

13 Lambda-Ausdrücke und funktionale Programmierung

13.1 Funktionale Schnittstellen und Lambda-Ausdrücke	807
13.1.1 Klassen implementieren Schnittstellen	807
13.1.2 Lambda-Ausdrücke implementieren Schnittstellen	809
13.1.3 Funktionale Schnittstellen	810
13.1.4 Der Typ eines Lambda-Ausdrucks ergibt sich durch den Zieltyp	811
13.1.5 Annotation @FunctionalInterface	816
13.1.6 Syntax für Lambda-Ausdrücke	817
13.1.7 Die Umgebung der Lambda-Ausdrücke und Variablenzugriffe	821
13.1.8 Ausnahmen in Lambda-Ausdrücken	827
13.1.9 Klassen mit einer abstrakten Methode als funktionale Schnittstelle? *	831
13.2 Methodenreferenz	831
13.2.1 Motivation	832
13.2.2 Methodenreferenzen mit ::	832
13.2.3 Varianten von Methodenreferenzen	833
13.3 Konstruktorreferenz	836
13.3.1 Konstruktorreferenzen schreiben	837
13.3.2 Parameterlose und parametrisierte Konstruktoren	838
13.3.3 Nützliche vordefinierte Schnittstellen für Konstruktorreferenzen	839
13.4 Funktionale Programmierung	839
13.4.1 Code = Daten	840

13.4.2	Programmierparadigmen: imperativ oder deklarativ	841
13.4.3	Das Wesen der funktionalen Programmierung	842
13.4.4	Funktionale Programmierung und funktionale Programmiersprachen	844
13.4.5	Funktionen höherer Ordnung am Beispiel von Comparator	846
13.4.6	Lambda-Ausdrücke als Abbildungen bzw. Funktionen betrachten	847
13.5	Funktionale Schnittstellen aus dem java.util.function-Paket	848
13.5.1	Blöcke mit Code und die funktionale Schnittstelle Consumer	849
13.5.2	Supplier	851
13.5.3	Prädikate und java.util.function.Predicate	851
13.5.4	Funktionen über die funktionale Schnittstelle java.util.function.Function	854
13.5.5	Ein bisschen Bi	858
13.5.6	Funktionale Schnittstellen mit Primitiven	861
13.6	Optional ist keine Nullnummer	863
13.6.1	Einsatz von null	864
13.6.2	Der Optional-Typ	866
13.6.3	Erst mal funktional mit Optional	868
13.6.4	Primitiv-Optionales mit speziellen Optional*-Klassen	871
13.7	Was ist jetzt so funktional?	874
13.7.1	Wiederverwertbarkeit	874
13.7.2	Zustandslos, immutable	875
13.8	Zum Weiterlesen	876

14 Architektur, Design und angewandte Objektorientierung

14.1	SOLIDe Modellierung	877
14.1.1	DRY, KISS und YAGNI	878
14.1.2	SOLID	878
14.1.3	Sei nicht STUPID	880
14.2	Architektur, Design und Implementierung	881
14.2.1	Implementierung und Idiome	881
14.2.2	Design	881
14.2.3	Architektur	882
14.3	Design-Patterns (Entwurfsmuster)	882
14.3.1	Motivation für Design-Patterns	883

14.3.2	Singleton	883
14.3.3	Fabrikmethoden	885
14.3.4	Das Beobachter-Pattern mit Listener realisieren	886
14.4	Zum Weiterlesen	890

15 Java Platform Module System

15.1	Klassenlader (Class Loader) und Modul-/Klassenpfad	891
15.1.1	Klassenladen auf Abruf	891
15.1.2	Klassenlader bei der Arbeit zusehen	892
15.1.3	JMOD-Dateien und JAR-Dateien	893
15.1.4	Woher die Klassen kommen: Suchorte und spezielle Klassenlader	894
15.1.5	Setzen des Suchpfads	895
15.2	Module im JPMS einbinden	897
15.2.1	Wer sieht wen?	898
15.2.2	Interne Plattformeigenschaften nutzen, --add-exports	899
15.2.3	Neue Module einbinden	901
15.3	Eigene Module entwickeln	902
15.3.1	Modul com.tutego.candytester	902
15.3.2	Moduldeklaration mit module-info.java und exports	903
15.3.3	Modul com.tutego.main	904
15.3.4	Modulinfodatei mit requires	904
15.3.5	Module mit den JVM-Optionen -p und -m ausführen	905
15.3.6	Modulinfodatei-Experimente	906
15.3.7	Automatische Module	907
15.3.8	Unbenanntes Modul	908
15.3.9	Lesbarkeit und Zugreifbarkeit	908
15.3.10	Modul-Migration	909
15.4	Zum Weiterlesen	911

16 Die Klassenbibliothek

16.1	Die Java-Klassenphilosophie	913
16.1.1	Modul, Paket, Typ	913
16.1.2	Übersicht über die Pakete der Standardbibliothek	915

16.2 Einfache Zeitmessung und Profiling *	917
16.2.1 Profiler *	918
16.3 Die Klasse Class	918
16.3.1 An ein Class-Objekt kommen	918
16.3.2 Eine Class ist ein Type	921
16.4 Die Utility-Klassen System und Properties	921
16.4.1 Speicher der JVM	922
16.4.2 Systemeigenschaften der Java-Umgebung	923
16.4.3 Eigene Properties von der Konsole aus setzen *	925
16.4.4 Zeilenumbruchzeichen, line.separator	927
16.4.5 Umgebungsvariablen des Betriebssystems	927
16.5 Sprachen der Länder	929
16.5.1 Sprachen in Regionen über Locale-Objekte	929
16.6 Wichtige Datum-Klassen im Überblick	933
16.6.1 Der 1.1.1970	934
16.6.2 System.currentTimeMillis()	934
16.6.3 Einfache Zeitumrechnungen durch TimeUnit	934
16.7 Date-Time-API	935
16.7.1 Erster Überblick	936
16.7.2 Menschenzeit und Maschinenzeit	937
16.7.3 Die Datumsklasse LocalDate	939
16.8 Logging mit Java	940
16.8.1 Logging-APIs	941
16.8.2 Logging mit java.util.logging	941
16.9 Maven: Build-Management und Abhängigkeiten auflösen	943
16.9.1 Dependency hinzunehmen	944
16.9.2 Lokales und das Remote Repository	945
16.9.3 Lebenszyklus, Phasen und Maven-Plugins	945
16.10 Zum Weiterlesen	945
17 Einführung in die nebenläufige Programmierung	947
<hr/>	
17.1 Nebenläufigkeit und Parallelität	947
17.1.1 Multitasking, Prozesse und Threads	948
17.1.2 Wie nebenläufige Programme die Geschwindigkeit steigern können	949
17.1.3 Java-Threads	950
17.1.4 Was Java für Nebenläufigkeit alles bietet	951

17.2 Existierende Threads und neue Threads erzeugen	951
17.2.1 Main-Thread	951
17.2.2 Wer bin ich?	952
17.2.3 Die Schnittstelle Runnable implementieren	952
17.2.4 Thread aufbauen	953
17.2.5 Thread mit Runnable starten	954
17.2.6 Runnable parametrisieren	956
17.2.7 Schläfer gesucht	957
17.2.8 Wann Threads fertig sind	958
17.2.9 Einen Thread höflich mit Interrupt beenden	959
17.2.10 Unbehandelte Ausnahmen, Thread-Ende und UncaughtExceptionHandler	961
17.2.11 Ein Rendezvous mit join(...) *	963
17.3 Der Ausführer (Executor) kommt ins Spiel	965
17.3.1 Die Schnittstelle Executor	966
17.3.2 Glücklich in der Gruppe – die Thread-Pools	967
17.3.3 Threads mit Rückgabe über Callable	970
17.3.4 Erinnerungen an die Zukunft – die Future-Rückgabe	972
17.3.5 Mehrere Callable-Objekte abarbeiten	975
17.3.6 CompletionService und ExecutorCompletionService	976
17.3.7 ScheduledExecutorService: wiederholende Aufgaben und Zeitsteuerungen	978
17.3.8 Asynchrones Programmieren mit CompletableFuture (CompletionStage)	978
17.4 Zum Weiterlesen	981

18 Einführung in Datenstrukturen und Algorithmen 983

18.1 Listen	983
18.1.1 Erstes Listen-Beispiel	984
18.1.2 Auswahlkriterium ArrayList oder LinkedList	985
18.1.3 Die Schnittstelle List	986
18.1.4 ArrayList	992
18.1.5 LinkedList	993
18.1.6 Der Array-Adapter Arrays.asList(..)	994
18.1.7 toArray(..) von Collection verstehen – die Gefahr einer Falle erkennen	996
18.1.8 Primitive Elemente in Datenstrukturen verwalten	998

18.2 Mengen (Sets)	998
18.2.1 Ein erstes Mengen-Beispiel	999
18.2.2 Methoden der Schnittstelle Set	1001
18.2.3 HashSet	1003
18.2.4 TreeSet – die sortierte Menge	1004
18.2.5 Die Schnittstellen NavigableSet und SortedSet	1006
18.2.6 LinkedHashSet	1008
18.3 Assoziative Speicher	1009
18.3.1 Die Klassen HashMap und TreeMap, statische Map-Methoden	1010
18.3.2 Einfügen und Abfragen des Assoziativspeichers	1013
18.4 Ausgewählte Basistypen	1015
18.5 Java-Stream-API	1016
18.5.1 Deklaratives Programmieren	1016
18.5.2 Interne versus externe Iteration	1017
18.5.3 Was ist ein Stream?	1018
18.6 Einen Stream erzeugen	1019
18.6.1 Stream.of*(...)	1021
18.6.2 Stream.generate(...) und Stream.iterate(..)	1021
18.6.3 Parallele oder sequenzielle Streams	1021
18.7 Terminale Operationen	1022
18.7.1 Die Anzahl der Elemente	1022
18.7.2 Und jetzt alle – forEach*(...)	1022
18.7.3 Einzelne Elemente aus dem Strom holen	1023
18.7.4 Existenztests mit Prädikaten	1024
18.7.5 Einen Strom auf sein kleinstes oder größtes Element reduzieren	1024
18.7.6 Einen Strom mit eigenen Funktionen reduzieren	1025
18.7.7 Stream-Elemente in eine Liste, ein Array oder einen Iterator übertragen	1026
18.7.8 Kollektoren	1027
18.8 Intermediäre Operationen	1029
18.8.1 Element-Vorschau	1029
18.8.2 Filtern von Elementen	1030
18.8.3 Statusbehaftete intermediäre Operationen	1030
18.8.4 Präfix-Operation	1032
18.8.5 Abbildungen	1033
18.8.6 Gatherer	1035
18.9 Zum Weiterlesen	1036

19 Einführung in grafische Oberflächen 1039

19.1 GUI-Frameworks	1039
19.1.1 Kommandozeile	1039
19.1.2 Grafische Benutzeroberfläche	1039
19.2 Abstract Window Toolkit (AWT)	1040
19.2.1 AWT-Frame	1041
19.2.2 Die paint(Graphics)-Methode für den AWT-Frame	1041
19.3 Java Foundation Classes und Swing	1043
19.3.1 Beispielanwendung in Swing	1043
19.4 JavaFX	1045
19.4.1 Fähigkeiten von JavaFX	1045
19.4.2 Die Geschichte von JavaFX: JavaFX 1, JavaFX 2, JavaFX 8, OpenJFX	1046
19.5 Standard Widget Toolkit (SWT) *	1047
19.5.1 Ursprung und Architektur	1047
19.5.2 Einsatz und Erweiterungen	1047
19.5.3 Bewertung und heutige Relevanz	1047
19.6 Zum Weiterlesen	1048

20 Einführung in Dateien und Datenströme 1049

20.1 Alte und neue Welt in java.io und java.nio	1049
20.1.1 java.io-Paket mit File-Klasse	1049
20.1.2 NIO.2 und das java.nio-Paket	1050
20.1.3 java.io.File oder java.nio.*-Typen?	1050
20.2 Dateisysteme und Pfade	1050
20.2.1 FileSystem und Path	1051
20.2.2 Die Utility-Klasse Files	1057
20.3 Dateien mit wahlfreiem Zugriff	1060
20.3.1 Ein RandomAccessFile zum Lesen und Schreiben öffnen	1060
20.3.2 Aus dem RandomAccessFile lesen	1061
20.3.3 Schreiben mit RandomAccessFile	1063
20.3.4 Die Länge des RandomAccessFile	1064
20.3.5 Hin und her in der Datei	1064
20.4 Basisklassen für die Ein-/Ausgabe	1065
20.4.1 Die vier abstrakten Basisklassen	1066

20.4.2	Die abstrakte Basisklasse OutputStream	1066
20.4.3	Die abstrakte Basisklasse InputStream	1069
20.4.4	Die abstrakte Basisklasse Writer	1071
20.4.5	Die Schnittstelle Appendable *	1072
20.4.6	Die abstrakte Basisklasse Reader	1072
20.4.7	Die Schnittstellen Closeable, AutoCloseable und Flushable	1076
20.5	Lesen aus Dateien und Schreiben in Dateien	1077
20.5.1	Byteorientierte Datenströme über Files beziehen	1078
20.5.2	Zeichenorientierte Datenströme über Files beziehen	1079
20.5.3	Die Funktion von OpenOption bei den Files.new*(...)-Methoden	1081
20.5.4	Ressourcen aus dem Klassenpfad und aus JAR-Dateien laden	1083
20.6	Zum Weiterlesen	1085

21 Einführung ins Datenbankmanagement mit JDBC 1087

21.1	Relationale Datenbanken und Java-Zugriffe	1087
21.1.1	Das relationale Modell	1087
21.1.2	Java-APIs zum Zugriff auf relationale Datenbanken	1088
21.1.3	Die JDBC-API und Implementierungen: JDBC-Treiber	1089
21.1.4	H2 ist das Werkzeug auf der Insel	1089
21.2	Eine Beispielabfrage	1090
21.2.1	Schritte zur Datenbankabfrage	1090
21.2.2	Mit Java auf die relationale Datenbank zugreifen	1090
21.3	Zum Weiterlesen	1091

22 Bits und Bytes, Mathematisches und Geld 1093

22.1	Bits und Bytes	1093
22.1.1	Die Bit-Operatoren Komplement, Und, Oder und XOR	1094
22.1.2	Repräsentation ganzer Zahlen in Java – das Zweierkomplement	1095
22.1.3	Das binäre (Basis 2), oktale (Basis 8), hexadezimale (Basis 16) Stellenwertsystem	1097
22.1.4	Auswirkung der Typumwandlung auf die Bit-Muster	1098
22.1.5	Vorzeichenlos arbeiten	1101
22.1.6	Die Verschiebeoperatoren	1103

22.1.7	Ein Bit setzen, löschen, umdrehen und testen	1106
22.1.8	Bit-Methoden der Integer- und Long-Klasse	1106
22.2	Gleitkomma-Arithmetik in Java	1108
22.2.1	Spezialwerte für Unendlich, Null, NaN	1109
22.2.2	Standardnotation und wissenschaftliche Notation bei Gleitkommazahlen *	1112
22.2.3	Mantisse und Exponent *	1112
22.3	Die Eigenschaften der Klasse Math	1114
22.3.1	Klassenvariablen der Klasse Math	1115
22.3.2	Absolutwerte und Vorzeichen	1115
22.3.3	Maximum/Minimum	1116
22.3.4	Runden von Werten	1117
22.3.5	Rest der ganzzahligen Division *	1119
22.3.6	Division mit Rundung in Richtung negativ unendlich, alternativer Restwert *	1120
22.3.7	Multiply-Accumulate	1122
22.3.8	Wurzel- und Exponentialmethoden	1122
22.3.9	Der Logarithmus *	1123
22.3.10	Winkelmethoden *	1124
22.3.11	Zufallszahlen	1125
22.4	Genauigkeit, Wertebereich eines Typs und Überlaufkontrolle *	1126
22.4.1	Der größte und der kleinste Wert	1126
22.4.2	Überlauf und alles ganz exakt	1127
22.4.3	Was bitte macht eine ulp?	1130
22.5	Zufallszahlen: Random, ThreadLocalRandom und SecureRandom	1131
22.5.1	Die Klasse Random	1131
22.5.2	Die Klasse ThreadLocalRandom	1135
22.5.3	Die Klasse SecureRandom *	1135
22.6	Große Zahlen *	1135
22.6.1	Die Klasse BigInteger	1136
22.6.2	Beispiel: ganz lange Fakultäten mit BigInteger	1143
22.6.3	Große Dezimalzahlen mit BigDecimal	1144
22.6.4	Mit MathContext komfortabel die Rechengenauigkeit setzen	1147
22.6.5	Noch schneller rechnen durch mutable Implementierungen	1148
22.7	Geld und Währung	1149
22.7.1	Geldbeträge repräsentieren	1149
22.7.2	ISO 4217	1149
22.7.3	Währungen in Java repräsentieren	1150
22.8	Zum Weiterlesen	1151

23 Testen mit JUnit	1153
23.1 Softwaretests	1153
23.1.1 Vorgehen beim Schreiben von Testfällen	1154
23.2 Das Test-Framework JUnit	1154
23.2.1 JUnit-Versionen	1155
23.2.2 JUnit aufnehmen	1155
23.2.3 Test-Driven Development und Test-First	1155
23.2.4 Testen, implementieren, testen, implementieren, testen, freuen	1157
23.2.5 JUnit-Tests ausführen	1159
23.2.6 assert*(...)-Methoden der Klasse Assertions	1159
23.2.7 Exceptions testen	1162
23.2.8 Grenzen für Ausführungszeiten festlegen	1163
23.2.9 Beschriftungen mit @DisplayName	1164
23.2.10 Verschachtelte Tests	1164
23.2.11 Tests ignorieren	1164
23.2.12 Mit Methoden der Assumptions-Klasse Tests abbrechen	1165
23.2.13 Parametrisierte Tests	1165
23.3 Java-Assertions-Bibliotheken und AssertJ	1167
23.3.1 AssertJ	1167
23.4 Aufbau größerer Testfälle	1169
23.4.1 Fixtures	1169
23.4.2 Sammlungen von Testklassen und Klassenorganisation	1171
23.5 Wie gutes Design das Testen ermöglicht	1171
23.6 Dummy, Fake, Stub und Mock	1173
23.6.1 Mockito	1174
23.7 JUnit-Erweiterungen, Testzusätze	1176
23.7.1 Webtests	1176
23.7.2 Tests der Datenbankschnittstelle	1176
23.8 Zum Weiterlesen	1177

24 Die Werkzeuge des JDK	1179
24.1 Übersicht	1179
24.1.1 Aufbau und gemeinsame Optionen	1180
24.2 Java-Quellen übersetzen	1180
24.2.1 Der Java-Compiler des JDK	1180
24.2.2 Native Compiler	1181
24.3 Die Java-Laufzeitumgebung	1182
24.3.1 Optionen der JVM	1182
24.3.2 Der Unterschied zwischen java.exe und javaw.exe	1184
24.4 Dokumentationskommentare mit Javadoc	1184
24.4.1 Einen Dokumentationskommentar setzen	1185
24.4.2 Markdown-Syntax und HTML-Tags in Dokumentationskommentaren *	1187
24.4.3 Mit dem Werkzeug javadoc eine Dokumentation erstellen	1188
24.4.4 Generierte Dateien	1188
24.4.5 Dokumentationskommentare im Überblick *	1189
24.4.6 Veraltete (deprecated) Typen und Eigenschaften	1190
24.4.7 Javadoc-Überprüfung mit DocLint	1193
24.4.8 Javadoc und Doclets *	1194
24.5 Das Archivformat JAR	1194
24.5.1 Das Dienstprogramm jar benutzen	1195
24.5.2 Das Manifest	1195
24.5.3 Applikationen in JAR-Archiven starten; ausführbare JAR-Dateien	1196
24.6 Zum Weiterlesen	1197
Anhang	1199
A Java SE-Module und Paketübersicht	1199
B Referenz Schlüsselwörter	1217
Index	1223