

Inhalt

Vorwort	31
1 Java ist auch eine Sprache	49
1.1 Historischer Hintergrund	49
1.2 Warum Java gut ist – die zentralen Eigenschaften	51
1.2.1 Bytecode	52
1.2.2 Ausführung des Bytecodes durch eine virtuelle Maschine	52
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	55
1.2.7 Java ist schnell – Optimierung und Just-in-time-Compilation	55
1.2.8 Das Java-Security-Modell	57
1.2.9 Zeiger und Referenzen	58
1.2.10 Bring den Müll raus, Garbage-Collector!	59
1.2.11 Ausnahmebehandlung	60
1.2.12 Angebot an Bibliotheken und Werkzeugen	61
1.2.13 Einfache Syntax der Programmiersprache Java	61
1.2.14 Java ist Open Source	63
1.2.15 Wofür sich Java weniger eignet	64
1.3 Java im Vergleich zu anderen Sprachen *	65
1.3.1 Java und C(++)	65
1.3.2 Java und JavaScript	66
1.3.3 Ein Wort zu Microsoft, Java und zu J++, J#	66
1.3.4 Java und C#/.NET	67
1.4 Weiterentwicklung und Verluste	68
1.4.1 Die Entwicklung von Java und seine Zukunftsaussichten	68
1.4.2 Features, Enhancements (Erweiterungen) und ein JSR	69
1.4.3 Applets	70
1.4.4 JavaFX	71
1.5 Java-Plattformen: Java SE, Jakarta EE, Java ME, Java Card	72
1.5.1 Die Java SE-Plattform	72
1.5.2 Java ME: Java für die Kleinen	74
1.5.3 Java für die ganz, ganz Kleinen	75

1.5.4	Java für die Großen: Java EE/Jakarta EE	75
1.5.5	Echtzeit-Java (Real-time Java)	76
1.6	Java SE-Implementierungen	77
1.6.1	OpenJDK	77
1.6.2	Oracle JDK	78
1.7	Die Installation des Oracle OpenJDK	81
1.7.1	OpenJDK unter Windows installieren	81
1.8	Das erste Programm compilieren und testen	83
1.8.1	Ein Quadratzahlen-Programm	84
1.8.2	Der Compilerlauf	85
1.8.3	Die Laufzeitumgebung	85
1.8.4	Häufige Compiler- und Interpreter-Probleme	86
1.9	Entwicklungsumgebungen im Allgemeinen	87
1.9.1	Eclipse IDE	87
1.9.2	IntelliJ IDEA	88
1.9.3	NetBeans	89
1.10	Eclipse IDE im Speziellen	89
1.10.1	Eclipse IDE entpacken und starten	91
1.10.2	Das erste Projekt anlegen	96
1.10.3	Verzeichnisstruktur für Java-Projekte *	98
1.10.4	Eine Klasse hinzufügen	99
1.10.5	Übersetzen und ausführen	100
1.10.6	Projekt einfügen, Workspace für die Programme wechseln	100
1.10.7	Plugins für Eclipse	101
1.11	Zum Weiterlesen	101

2 Imperative Sprachkonzepte 103

2.1	Elemente der Programmiersprache Java	103
2.1.1	Token	103
2.1.2	Textkodierung durch Unicode-Zeichen	104
2.1.3	Bezeichner	104
2.1.4	Literale	107
2.1.5	(Reservierte) Schlüsselwörter	107
2.1.6	Zusammenfassung der lexikalischen Analyse	108
2.1.7	Kommentare	109
2.2	Von der Klasse zur Anweisung	111
2.2.1	Was sind Anweisungen?	111

2.2.2	Klassendeklaration	112
2.2.3	Die Reise beginnt am main(String[])	113
2.2.4	Der erste Methodenaufruf: println(...)	113
2.2.5	Atomare Anweisungen und Anweisungssequenzen	114
2.2.6	Mehr zu print(...), println(...) und printf(...) für Bildschirmausgaben	115
2.2.7	Die API-Dokumentation	117
2.2.8	Ausdrücke	118
2.2.9	Ausdrucksanweisung	119
2.2.10	Erste Idee der Objektorientierung	120
2.2.11	Modifizierer	121
2.2.12	Gruppieren von Anweisungen mit Blöcken	121
2.3	Datentypen, Typisierung, Variablen und Zuweisungen	122
2.3.1	Primitive Datentypen im Überblick	125
2.3.2	Variablendeklarationen	127
2.3.3	Automatisches Feststellen der Typen mit var	130
2.3.4	Konsoleneingaben	131
2.3.5	Fließkommazahlen mit den Datentypen float und double	133
2.3.6	Ganzzahlige Datentypen	135
2.3.7	Wahrheitswerte	137
2.3.8	Unterstriche in Zahlen *	137
2.3.9	Alphanumerische Zeichen	138
2.3.10	Gute Namen, schlechte Namen	138
2.3.11	Initialisierung von lokalen Variablen	139
2.4	Ausdrücke, Operanden und Operatoren	140
2.4.1	Zuweisungsoperator	141
2.4.2	Arithmetische Operatoren	142
2.4.3	Unäres Minus und Plus	146
2.4.4	Präfix- oder Postfix-Inkrement und -Dekrement	147
2.4.5	Zuweisung mit Operation (Verbundoperator)	149
2.4.6	Die relationalen Operatoren und die Gleichheitsoperatoren	150
2.4.7	Logische Operatoren: Nicht, Und, Oder, XOR	152
2.4.8	Kurzschluss-Operatoren	153
2.4.9	Der Rang der Operatoren in der Auswertungsreihenfolge	155
2.4.10	Die Typumwandlung (das Casting)	158
2.4.11	Überladenes Plus für Strings	164
2.4.12	Operator vermisst *	165
2.5	Bedingte Anweisungen oder Fallunterscheidungen	166
2.5.1	Verzweigung mit der if-Anweisung	166
2.5.2	Die Alternative mit einer if-else-Anweisung wählen	169
2.5.3	Der Bedingungsoperator	173
2.5.4	Die switch-Anweisung bietet die Alternative	175

2.6	Immer das Gleiche mit den Schleifen	182
2.6.1	Die while-Schleife	182
2.6.2	Die do-while-Schleife	184
2.6.3	Die for-Schleife	186
2.6.4	Schleifenbedingungen und Vergleiche mit == *	190
2.6.5	Schleifenabbruch mit break und zurück zum Test mit continue	192
2.6.6	break und continue mit Marken *	195
2.7	Methoden einer Klasse	199
2.7.1	Bestandteil einer Methode	200
2.7.2	Signatur-Beschreibung in der Java-API	201
2.7.3	Aufruf einer Methode	203
2.7.4	Methoden ohne Parameter deklarieren	203
2.7.5	Statische Methoden (Klassenmethoden)	204
2.7.6	Parameter, Argument und Wertübergabe	205
2.7.7	Methoden vorzeitig mit return beenden	207
2.7.8	Nicht erreichbarer Quellcode bei Methoden *	208
2.7.9	Methoden mit Rückgaben	209
2.7.10	Methoden überladen	214
2.7.11	Gültigkeitsbereich	216
2.7.12	Vorgegebener Wert für nicht aufgeführte Argumente *	217
2.7.13	Finale lokale Variablen	218
2.7.14	Rekursive Methoden *	219
2.7.15	Die Türme von Hanoi *	224
2.8	Zum Weiterlesen	226

3 Klassen und Objekte 227

3.1	Objektorientierte Programmierung (OOP)	227
3.1.1	Warum überhaupt OOP?	227
3.1.2	Denk ich an Java, denk ich an Wiederverwendbarkeit	228
3.2	Eigenschaften einer Klasse	229
3.2.1	Klassenarbeit mit Point	230
3.3	Natürlich modellieren mit der UML (Unified Modeling Language) *	230
3.3.1	Hintergrund und Geschichte der UML *	231
3.3.2	Wichtige Diagrammtypen der UML *	232
3.3.3	UML-Werkzeuge *	233
3.4	Neue Objekte erzeugen	234
3.4.1	Ein Exemplar einer Klasse mit dem Schlüsselwort new anlegen	235

3.4.2	Der Zusammenhang von new, Heap und Garbage-Collector	235
3.4.3	Deklariere von Referenzvariablen	236
3.4.4	Jetzt mach mal 'nen Punkt: Zugriff auf Objektattribute und -methoden	238
3.4.5	Überblick über Point-Methoden	242
3.4.6	Konstrukteure nutzen	245
3.5	ZZZZZnake	246
3.6	Pakete schnüren, Importe und Kompilationseinheiten	249
3.6.1	Java-Pakete	249
3.6.2	Pakete der Standardbibliothek	249
3.6.3	Volle Qualifizierung und import-Deklaration	249
3.6.4	Mit import p1.p2.* alle Typen eines Pakets erreichen	251
3.6.5	Hierarchische Strukturen über Pakete	251
3.6.6	Die package-Deklaration	252
3.6.7	Unbenanntes Paket (default package)	253
3.6.8	Klassen mit gleichen Namen in unterschiedlichen Paketen *	254
3.6.9	Kompilationseinheit (Compilation Unit)	254
3.6.10	Statischer Import *	255
3.7	Mit Referenzen arbeiten, Identität und Gleichheit (Gleichwertigkeit)	256
3.7.1	null-Referenz und die Frage der Philosophie	256
3.7.2	Alles auf null? Referenzen testen	259
3.7.3	Zuweisungen bei Referenzen	260
3.7.4	Methoden mit Referenztypen als Parametern	261
3.7.5	Identität von Objekten	265
3.7.6	Gleichheit (Gleichwertigkeit) und die Methode equals(...)	266
3.8	Zum Weiterlesen	268
4	Arrays und ihre Anwendungen	269
<hr/>		
4.1	Arrays	269
4.1.1	Grundbestandteile	269
4.1.2	Deklaration von Array-Variablen	270
4.1.3	Array-Objekte mit new erzeugen	271
4.1.4	Arrays mit Inhalt	272
4.1.5	Die Länge eines Arrays über das Attribut length auslesen	273
4.1.6	Zugriff auf die Elemente über den Index	274
4.1.7	Typische Array-Fehler	276
4.1.8	Arrays als Methodenparameter	277
4.1.9	Vorinitialisierte Arrays	278
4.1.10	Die erweiterte for-Schleife	279

4.1.11	Arrays mit nichtprimitiven Elementen	281
4.1.12	Methode mit variabler Argumentanzahl (Varargs)	283
4.1.13	Mehrdimensionale Arrays *	285
4.1.14	Nichtrechteckige Arrays *	289
4.1.15	Die Wahrheit über die Array-Initialisierung *	291
4.1.16	Mehrere Rückgabewerte *	292
4.1.17	Klonen kann sich lohnen – Arrays vermehren *	293
4.1.18	Array-Inhalte kopieren *	294
4.1.19	Die Klasse Arrays zum Vergleichen, Füllen, Suchen, Sortieren nutzen	295
4.1.20	Eine lange Schlange	308
4.2	Der Einstiegspunkt für das Laufzeitsystem: main(...)	311
4.2.1	Korrekte Deklaration der Startmethode	311
4.2.2	Kommandozeilenargumente verarbeiten	312
4.2.3	Der Rückgabotyp von main(...) und System.exit(int) *	313
4.3	Zum Weiterlesen	315

5 Der Umgang mit Zeichenketten 317

5.1	Von ASCII über ISO-8859-1 zu Unicode	317
5.1.1	ASCII	317
5.1.2	ISO/IEC 8859-1	318
5.1.3	Unicode	319
5.1.4	Unicode-Zeichenkodierung	321
5.1.5	Escape-Sequenzen/Fluchtsymbole	322
5.1.6	Schreibweise für Unicode-Zeichen und Unicode-Escapes	323
5.1.7	Java-Versionen gehen mit Unicode-Standard Hand in Hand *	325
5.2	Die Character-Klasse	326
5.2.1	Ist das so?	326
5.2.2	Zeichen in Großbuchstaben/Kleinbuchstaben konvertieren	329
5.2.3	Vom Zeichen zum String	330
5.2.4	Von char in int: vom Zeichen zur Zahl *	330
5.3	Zeichenfolgen	332
5.4	Die Klasse String und ihre Methoden	334
5.4.1	String-Literale als String-Objekte für konstante Zeichenketten	334
5.4.2	Konkatenation mit +	334
5.4.3	String-Länge und Test auf Leer-String	335
5.4.4	Zugriff auf ein bestimmtes Zeichen mit charAt(int)	336
5.4.5	Nach enthaltenen Zeichen und Zeichenfolgen suchen	337

5.4.6	Das Hangman-Spiel	340
5.4.7	Gut, dass wir verglichen haben	342
5.4.8	String-Teile extrahieren	346
5.4.9	Strings anhängen, zusammenfügen, Groß-/Kleinschreibung und Weißraum	350
5.4.10	Gesucht, gefunden, ersetzt	354
5.4.11	String-Objekte mit Konstruktoren und aus Wiederholungen erzeugen *	356
5.5	Veränderbare Zeichenketten mit StringBuilder und StringBuffer	360
5.5.1	Anlegen von StringBuilder-Objekten	361
5.5.2	StringBuilder in andere Zeichenkettenformate konvertieren	362
5.5.3	Zeichen(folgen) erfragen	362
5.5.4	Daten anhängen	362
5.5.5	Zeichen(folgen) setzen, löschen und umdrehen	364
5.5.6	Länge und Kapazität eines StringBuilder-Objekts *	367
5.5.7	Vergleich von StringBuilder-Exemplaren und String mit StringBuilder	368
5.5.8	hashCode() bei StringBuilder *	370
5.6	CharSequence als Basistyp	370
5.7	Konvertieren zwischen Primitiven und Strings	373
5.7.1	Unterschiedliche Typen in String-Repräsentationen konvertieren	373
5.7.2	String-Inhalt in einen primitiven Wert konvertieren	375
5.7.3	String-Repräsentation im Format Binär, Hex, Oktal *	377
5.7.4	parseXXX(...) und printXXX()-Methoden in DatatypeConverter *	381
5.8	Strings zusammenhängen (konkateneren)	382
5.8.1	Strings mit StringJoiner zusammenhängen	383
5.9	Zerlegen von Zeichenketten	384
5.9.1	Splitten von Zeichenketten mit split(...)	385
5.9.2	Yes we can, yes we scan – die Klasse Scanner	386
5.10	Ausgaben formatieren	390
5.10.1	Formatieren und Ausgeben mit format()	390
5.11	Zum Weiterlesen	396
6	Eigene Klassen schreiben	397
<hr/>		
6.1	Eigene Klassen mit Eigenschaften deklarieren	397
6.1.1	Attribute deklarieren	398
6.1.2	Methoden deklarieren	400
6.1.3	Verdeckte (shadowed) Variablen	404
6.1.4	Die this-Referenz	405

6.2	Privatsphäre und Sichtbarkeit	409
6.2.1	Für die Öffentlichkeit: public	409
6.2.2	Kein Public Viewing – Passwörter sind privat	410
6.2.3	Wieso nicht freie Methoden und Variablen für alle?	411
6.2.4	Privat ist nicht ganz privat: Es kommt darauf an, wer’s sieht *	412
6.2.5	Zugriffsmethoden für Attribute deklarieren	412
6.2.6	Setter und Getter nach der JavaBeans-Spezifikation	413
6.2.7	Paketsichtbar	415
6.2.8	Zusammenfassung zur Sichtbarkeit	417
6.3	Eine für alle – statische Methoden und statische Attribute	419
6.3.1	Warum statische Eigenschaften sinnvoll sind	420
6.3.2	Statische Eigenschaften mit static	421
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 *	423
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	Typunsichere Aufzählungen	427
6.4.3	Aufzählungstypen: typsichere Aufzählungen mit enum	429
6.5	Objekte anlegen und zerstören	434
6.5.1	Konstruktoren schreiben	434
6.5.2	Verwandschaft von Methode und Konstruktor	436
6.5.3	Der Standard-Konstruktor (default constructor)	436
6.5.4	Parametrisierte und überladene Konstruktoren	438
6.5.5	Copy-Konstruktor	440
6.5.6	Einen anderen Konstruktor der gleichen Klasse mit this(...) aufrufen	441
6.5.7	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

7.1 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	464
7.2.1 Vererbung in Java	464
7.2.2 Spielobjekte modellieren	465
7.2.3 Die implizite Basisklasse java.lang.Object	467
7.2.4 Einfach- und Mehrfachvererbung *	467
7.2.5 Sehen Kinder alles? Die Sichtbarkeit protected	468
7.2.6 Konstruktoren in der Vererbung und super(...)	469
7.3 Typen in Hierarchien	475
7.3.1 Automatische und explizite Typumwandlung	475
7.3.2 Das Substitutionsprinzip	478
7.3.3 Typen mit dem instanceof-Operator testen	480
7.4 Methoden überschreiben	482
7.4.1 Methoden in Unterklassen mit neuem Verhalten ausstatten	482
7.4.2 Mit super an die Eltern	487
7.4.3 Finale Klassen und finale Methoden	489
7.4.4 Kovariante Rückgabetypen	491
7.4.5 Array-Typen und Kovarianz *	492
7.5 Drum prüfe, wer sich dynamisch bindet	493
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	510
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	515
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 *	543
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	Eine NumberFormatException auffangen	554
8.1.3	Bitte nicht schlucken – leere catch-Blöcke	558
8.1.4	Wiederholung abgebrochener Bereiche *	558
8.1.5	Mehrere Ausnahmen auffangen	559
8.1.6	Ablauf einer Ausnahmesituation	562
8.1.7	throws im Methodenkopf angeben	562
8.1.8	Abschlussbehandlung mit finally	564
8.2	Die Klassenhierarchie der Ausnahmen	569
8.2.1	Eigenschaften des Exception-Objekts	569
8.2.2	Basistyp Throwable	570

8.2.3	Die Exception-Hierarchie	571
8.2.4	Oberausnahmen auffangen	572
8.2.5	Schon gefangen?	574
8.2.6	Alles geht als Exception durch	574
8.2.7	Zusammenfassen gleicher catch-Blöcke mit dem multi-catch	576
8.3	RuntimeException muss nicht aufgefangen werden	579
8.3.1	Beispiele für RuntimeException-Klassen	580
8.3.2	Kann man abfangen, muss man aber nicht	581
8.4	Harte Fehler – Error *	581
8.5	Auslösen eigener Exceptions	582
8.5.1	Mit throw Ausnahmen auslösen	582
8.5.2	Vorhandene Runtime-Ausnahmetypen kennen und nutzen	584
8.5.3	Parameter testen und gute Fehlermeldungen	587
8.5.4	Neue Exception-Klassen deklarieren	589
8.5.5	Eigene Ausnahmen als Unterklassen von Exception oder RuntimeException?	590
8.5.6	Ausnahmen abfangen und weiterleiten *	593
8.5.7	Aufruf-Stack von Ausnahmen verändern *	594
8.5.8	Präzises rethrow *	595
8.5.9	Geschachtelte Ausnahmen *	599
8.6	Automatisches Ressourcen-Management (try mit Ressourcen)	602
8.6.1	try mit Ressourcen	602
8.6.2	Die Schnittstelle AutoCloseable	605
8.6.3	Mehrere Ressourcen nutzen	607
8.6.4	try mit Ressourcen auf null-Ressourcen	608
8.6.5	Unterdrückte Ausnahmen *	608
8.7	Besonderheiten bei der Ausnahmebehandlung *	611
8.7.1	Rückgabewerte bei ausgelösten Ausnahmen	612
8.7.2	Ausnahmen und Rückgaben verschwinden – das Duo return und finally	612
8.7.3	throws bei überschriebenen Methoden	613
8.7.4	Nicht erreichbare catch-Klauseln	615
8.8	Assertions *	617
8.8.1	Assertions in eigenen Programmen nutzen	617
8.8.2	Assertions aktivieren und Laufzeit-Errors	617
8.8.3	Assertions feiner aktivieren oder deaktivieren	620
8.9	Zum Weiterlesen	621

9 Geschachtelte Typen 623

9.1	Geschachtelte Klassen, Schnittstellen, 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	Objektgleichheit 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 *	668
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 Programmargumente 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	687
10.5.3 Von einer String-Repräsentation parsen	688
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	694
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	708
10.6.5 Interne Iteration	708
10.6.6 Einen eigenen Iterable implementieren *	709
10.7 Die Spezial-Oberklasse Enum	710
10.7.1 Methoden auf Enum-Objekten	711
10.7.2 Aufzählungen mit eigenen Methoden und Initialisierern *	714
10.7.3 enum mit eigenen Konstruktoren *	717

10.8 Annotationen in der Java SE	720
10.8.1 Orte für Annotationen	721
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 Einschränken der Typen über Bounds	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>> *	771

11.6	Konsequenzen der Typlöschung: Typ-Token, Arrays und Brücken *	773
11.6.1	Typ-Token	773
11.6.2	Super-Type-Token	775
11.6.3	Generics und Arrays	776
11.6.4	Brückenmethoden	777
11.7	Zum Weiterlesen	783

12 Lambda-Ausdrücke und funktionale Programmierung

785

12.1	Code = Daten	785
12.2	Funktionale Schnittstellen und Lambda-Ausdrücke im Detail	788
12.2.1	Funktionale Schnittstellen	789
12.2.2	Typ eines Lambda-Ausdrucks ergibt sich durch Zieltyp	790
12.2.3	Annotation @FunctionalInterface	794
12.2.4	Syntax für Lambda-Ausdrücke	795
12.2.5	Die Umgebung der Lambda-Ausdrücke und Variablenzugriffe	800
12.2.6	Ausnahmen in Lambda-Ausdrücken	804
12.2.7	Klassen mit einer abstrakten Methode als funktionale Schnittstelle? *	808
12.3	Methodenreferenz	809
12.3.1	Varianten von Methodenreferenzen	811
12.4	Konstruktorreferenz	813
12.4.1	Parameterlose und parametrisierte Konstruktoren	815
12.4.2	Nützliche vordefinierte Schnittstellen für Konstruktorreferenzen	815
12.5	Implementierung von Lambda-Ausdrücken *	816
12.6	Funktionale Programmierung mit Java	817
12.6.1	Programmierparadigmen: imperativ oder deklarativ	817
12.6.2	Funktionale Programmierung und funktionale Programmiersprachen	818
12.6.3	Funktionale Programmierung in Java am Beispiel vom Comparator	819
12.6.4	Lambda-Ausdrücke als Funktionen sehen	820
12.7	Funktionale Schnittstelle aus dem java.util.function-Paket	821
12.7.1	Blöcke mit Code und die funktionale Schnittstelle Consumer	822
12.7.2	Supplier	824
12.7.3	Prädikate und java.util.function.Predicate	824
12.7.4	Funktionen über die funktionale Schnittstelle java.util.function.Function	826
12.7.5	Ein bisschen Bi ...	830
12.7.6	Funktionale Schnittstellen mit Primitiven	833

12.8	Optional ist keine Nullnummer	836
12.8.1	Optional-Typ	838
12.8.2	Primitive optionale Typen	841
12.8.3	Erstmal funktional mit Optional	842
12.9	Was ist jetzt so funktional?	847
12.10	Zum Weiterlesen	849

13 Architektur, Design und angewandte Objektorientierung 851

13.1	Architektur, Design und Implementierung	851
13.2	Design-Patterns (Entwurfsmuster)	852
13.2.1	Motivation für Design-Patterns	852
13.2.2	Singleton	853
13.2.3	Fabrikmethoden	854
13.2.4	Das Beobachter-Pattern mit Listener realisieren	856
13.3	Zum Weiterlesen	860

14 Komponenten, JavaBeans und Module 861

14.1	JavaBeans	861
14.1.1	Properties (Eigenschaften)	862
14.1.2	Einfache Eigenschaften	863
14.1.3	Indizierte Eigenschaften	863
14.1.4	Gebundene Eigenschaften und PropertyChangeListener	863
14.1.5	Veto-Eigenschaften – dagegen!	867
14.2	Klassenslader (Class Loader) und Modul-/Klassenpfad	870
14.2.1	Klassenslader auf Abruf	871
14.2.2	Klassenslader bei der Arbeit zusehen	871
14.2.3	JMOD-Dateien und JAR-Dateien	872
14.2.4	Woher die kleinen Klassen kommen: die Suchorte und spezielle Klassenslader	873
14.2.5	Setzen des Modulpfades	874
14.3	Module entwickeln und einbinden	876
14.3.1	Wer sieht wen	876
14.3.2	Plattform-Module und JMOD-Beispiel	878

14.3.3	Interne Plattformeigenschaften nutzen, --add-exports	878
14.3.4	Neue Module einbinden, --add-modules und --add-opens	881
14.3.5	Projektabhängigkeiten in Eclipse	882
14.3.6	Benannte Module und module-info.java	884
14.3.7	Automatische Module	887
14.3.8	Unbenanntes Modul	888
14.3.9	Lesbarkeit und Zugreifbarkeit	889
14.3.10	Modul-Migration	890
14.4	Zum Weiterlesen	891

15 Die Klassenbibliothek 893

15.1	Die Java-Klassenphilosophie	893
15.1.1	Modul, Paket, Typ	893
15.1.2	Übersicht über die Pakete der Standardbibliothek	896
15.2	Einfache Zeitmessung und Profiling *	901
15.3	Die Klasse Class	904
15.3.1	An ein Class-Objekt kommen	904
15.3.2	Eine Class ist ein Type	906
15.4	Klassenlader	907
15.4.1	Die Klasse java.lang.ClassLoader	908
15.5	Die Utility-Klassen System und Properties	908
15.5.1	Speicher der JVM	910
15.5.2	Anzahl CPUs/Kerne	911
15.5.3	Systemeigenschaften der Java-Umgebung	911
15.5.4	Eigene Properties von der Konsole aus setzen *	913
15.5.5	Zeilenumbruchzeichen, line.separator	915
15.5.6	Umgebungsvariablen des Betriebssystems	916
15.6	Sprachen der Länder	917
15.6.1	Sprachen in Regionen über Locale-Objekte	917
15.7	Wichtige Datum-Klassen im Überblick	922
15.7.1	Der 1.1.1970	922
15.7.2	System.currentTimeMillis()	923
15.7.3	Einfache Zeitumrechnungen durch TimeUnit	923
15.8	Date-Time-API	924
15.8.1	Menschenzeit und Maschinenzeit	926
15.8.2	Datumsklasse LocalDate	929

15.9 Logging mit Java	930
15.9.1 Logging-APIs	930
15.9.2 Logging mit java.util.logging	931
15.10 Maven: Build-Management und Abhängigkeiten auflösen	933
15.10.1 Beispielprojekt in Eclipse mit Maven	934
15.10.2 Properties hinzunehmen	934
15.10.3 Dependency hinzunehmen	935
15.10.4 Lokales und Remote-Repository	936
15.10.5 Lebenszyklus, Phasen und Maven-Plugins	936
15.10.6 Archetypes	937
15.11 Zum Weiterlesen	937

16 Einführung in die nebenläufige Programmierung 939

16.1 Nebenläufigkeit und Parallelität	939
16.1.1 Multitasking, Prozesse, Threads	940
16.1.2 Threads und Prozesse	940
16.1.3 Wie nebenläufige Programme die Geschwindigkeit steigern können	941
16.1.4 Was Java für Nebenläufigkeit alles bietet	943
16.2 Laufende Threads, neue Threads erzeugen	943
16.2.1 Main-Thread	944
16.2.2 Wer bin ich?	944
16.2.3 Die Schnittstelle Runnable implementieren	944
16.2.4 Thread mit Runnable starten	946
16.2.5 Runnable parametrisieren	947
16.2.6 Die Klasse Thread erweitern	948
16.3 Thread-Eigenschaften und Zustände	950
16.3.1 Der Name eines Threads	951
16.3.2 Die Zustände eines Threads *	951
16.3.3 Schläfer gesucht	952
16.3.4 Mit yield() und onSpinWait() auf Rechenzeit verzichten	954
16.3.5 Wann Threads fertig sind	955
16.3.6 Einen Thread höflich mit Interrupt beenden	955
16.3.7 Unbehandelte Ausnahmen, Thread-Ende und UncaughtExceptionHandler	958
16.3.8 Der stop() von außen und die Rettung mit ThreadDeath *	959
16.3.9 Ein Rendezvous mit join(...) *	960
16.3.10 Arbeit niederlegen und wieder aufnehmen *	962
16.3.11 Priorität *	963

16.4	Der Ausführer (Executor) kommt	964
16.4.1	Die Schnittstelle Executor	965
16.4.2	Glücklich in der Gruppe – die Thread-Pools	966
16.4.3	Threads mit Rückgabe über Callable	968
16.4.4	Erinnerungen an die Zukunft – die Future-Rückgabe	970
16.4.5	Mehrere Callable-Objekte abarbeiten	973
16.4.6	CompletionService und ExecutorCompletionService	975
16.4.7	ScheduledExecutorService für wiederholende Aufgaben und Zeitsteuerungen nutzen	976
16.5	Zum Weiterlesen	977

17 Einführung in Datenstrukturen und Algorithmen 979

17.1	Listen	979
17.1.1	Erstes Listen-Beispiel	980
17.1.2	Auswahlkriterium ArrayList oder LinkedList	981
17.1.3	Die Schnittstelle List	981
17.1.4	ArrayList	988
17.1.5	LinkedList	989
17.1.6	Der Array-Adapter Arrays.asList(...)	991
17.1.7	ListIterator *	993
17.1.8	toArray(...) von Collection verstehen – die Gefahr einer Falle erkennen	994
17.1.9	Primitive Elemente in Datenstrukturen verwalten	998
17.2	Mengen (Sets)	998
17.2.1	Ein erstes Mengen-Beispiel	999
17.2.2	Methoden der Schnittstelle Set	1001
17.2.3	HashSet	1003
17.2.4	TreeSet – die sortierte Menge	1003
17.2.5	Die Schnittstellen NavigableSet und SortedSet	1005
17.2.6	LinkedHashSet	1008
17.3	Java Stream API	1009
17.3.1	Deklaratives Programmieren	1009
17.3.2	Interne versus externe Iteration	1009
17.3.3	Was ist ein Stream?	1010
17.4	Stream erzeugen	1012
17.4.1	Parallele oder sequenzielle Streams	1015
17.5	Terminale Stream-Operationen	1016
17.5.1	Anzahl Elemente	1016

17.5.2	Und jetzt alle – forEachXXX(...)	1017
17.5.3	Einzelne Elemente aus dem Strom holen	1017
17.5.4	Existenztests mit Prädikaten	1018
17.5.5	Strom reduzieren auf kleinstes/größtes Element	1018
17.5.6	Strom mit eigenen Funktionen reduzieren	1019
17.5.7	Ergebnisse in einen Container schreiben, Teil 1: collect(...)	1021
17.5.8	Ergebnisse in einen Container schreiben, Teil 2: Collector und Collectors	1022
17.5.9	Ergebnisse in einen Container schreiben, Teil 3: Gruppierungen	1024
17.5.10	Stream-Elemente in Array oder Iterator übertragen	1026
17.6	Intermediäre Stream-Operationen	1027
17.6.1	Element-Vorschau	1028
17.6.2	Filtern von Elementen	1028
17.6.3	Statusbehaftete intermediäre Operationen	1028
17.6.4	Präfix-Operation	1030
17.6.5	Abbildungen	1031
17.7	Zum Weiterlesen	1033

18 Einführung in grafische Oberflächen 1035

18.1	GUI-Frameworks	1035
18.1.1	Kommandozeile	1035
18.1.2	Grafische Benutzeroberfläche	1035
18.1.3	Abstract Window Toolkit (AWT)	1036
18.1.4	Java Foundation Classes und Swing	1036
18.1.5	JavaFX	1036
18.1.6	SWT (Standard Widget Toolkit) *	1038
18.2	Deklarative und programmierte Oberflächen	1039
18.2.1	GUI-Beschreibungen in JavaFX	1040
18.2.2	Deklarative GUI-Beschreibungen für Swing?	1040
18.3	GUI-Builder	1041
18.3.1	GUI-Builder für JavaFX	1041
18.3.2	GUI-Builder für Swing	1042
18.4	Mit dem Eclipse WindowBuilder zur ersten Swing-Oberfläche	1042
18.4.1	WindowBuilder installieren	1042
18.4.2	Mit WindowBuilder eine GUI-Klasse hinzufügen	1044
18.4.3	Layoutprogramm starten	1046
18.4.4	Grafische Oberfläche aufbauen	1047
18.4.5	Swing-Komponenten-Klassen	1049
18.4.6	Funktionalität geben	1051

18.5	Grundlegendes zum Zeichnen	1054
18.5.1	Die paint(Graphics)-Methode für das AWT-Frame	1054
18.5.2	Die ereignisorientierte Programmierung ändert Fensterinhalte	1056
18.5.3	Zeichnen von Inhalten auf einen JFrame	1057
18.5.4	Auffordern zum Neuzeichnen mit repaint(...)	1058
18.5.5	Java 2D-API	1059
18.6	Zum Weiterlesen	1060

19 Einführung in Dateien und Datenströme 1061

19.1	Alte und neue Welt in java.io und java.nio	1061
19.1.1	java.io-Paket mit File-Klasse	1061
19.1.2	NIO.2 und java.nio-Paket	1062
19.1.3	java.io.File oder java.nio.*?	1062
19.2	Dateisysteme und Pfade	1063
19.2.1	FileSystem und Path	1063
19.2.2	Die Utility-Klasse Files	1069
19.3	Dateien mit wahlfreiem Zugriff	1072
19.3.1	Ein RandomAccessFile zum Lesen und Schreiben öffnen	1072
19.3.2	Aus dem RandomAccessFile lesen	1073
19.3.3	Schreiben mit RandomAccessFile	1076
19.3.4	Die Länge des RandomAccessFile	1076
19.3.5	Hin und her in der Datei	1077
19.4	Basisklassen für die Ein-/Ausgabe	1078
19.4.1	Die vier abstrakten Basisklassen	1078
19.4.2	Die abstrakte Basisklasse OutputStream	1079
19.4.3	Die abstrakte Basisklasse InputStream	1081
19.4.4	Die abstrakte Basisklasse Writer	1084
19.4.5	Die Schnittstelle Appendable *	1086
19.4.6	Die abstrakte Basisklasse Reader	1087
19.4.7	Die Schnittstellen Closeable, AutoCloseable und Flushable	1090
19.5	Lesen aus Dateien und Schreiben in Dateien	1092
19.5.1	Byteorientierte Datenströme über Files beziehen	1092
19.5.2	Zeichenorientierte Datenströme über Files beziehen	1093
19.5.3	Funktion von OpenOption bei den Files.newXXX(...)-Methoden	1095
19.5.4	Ressourcen aus dem Modulpfad und aus JAR-Dateien laden	1096
19.6	Zum Weiterlesen	1098

20 Einführung ins Datenbankmanagement mit JDBC 1099

20.1 Relationale Datenbanken und Datenbankmanagementsysteme	1099
20.1.1 Das relationale Modell	1099
20.1.2 H2-Datenbank	1100
20.1.3 Weitere Datenbank-Management-Systeme *	1100
20.2 JDBC und Datenbanktreiber	1102
20.2.1 H2-Datenbank und JDBC-Treiber	1103
20.2.2 JDBC-Versionen *	1103
20.3 Eine Beispielabfrage	1104
20.3.1 Schritte zur Datenbankabfrage	1104
20.3.2 Mit Java auf die relationale Datenbank zugreifen	1105
20.4 Zum Weiterlesen	1106

21 Einführung in <XML> 1107

21.1 Auszeichnungssprachen	1107
21.1.1 Extensible Markup Language (XML)	1108
21.2 Eigenschaften von XML-Dokumenten	1108
21.2.1 Elemente und Attribute	1108
21.2.2 Beschreibungssprache für den Aufbau von XML-Dokumenten	1111
21.2.3 Schema – die moderne Alternative zu DTD	1115
21.2.4 Namensraum (Namespace)	1118
21.2.5 XML-Applikationen *	1119
21.3 Die Java-APIs für XML	1120
21.3.1 Das Document Object Model (DOM)	1121
21.3.2 Pull-API StAX	1121
21.3.3 Simple API for XML Parsing (SAX)	1121
21.3.4 Java Document Object Model (JDOM)	1122
21.3.5 JAXP als Java-Schnittstelle zu XML	1122
21.3.6 DOM-Bäume einlesen mit JAXP *	1123
21.4 Java Architecture for XML Binding (JAXB)	1124
21.4.1 JAXB raus aus der Java SE, Abhängigkeit rein in die POM	1124
21.4.2 Bean für JAXB aufbauen	1125
21.4.3 Utility-Klasse JAXB	1126

21.4.4	Ganze Objektgraphen schreiben und lesen	1127
21.4.5	JAXBContext und Marshaller/Unmarshaller nutzen	1129
21.5	Zum Weiterlesen	1131

22 Bits und Bytes, Mathematisches und Geld 1133

22.1	Bits und Bytes	1133
22.1.1	Die Bit-Operatoren Komplement, Und, Oder und XOR	1134
22.1.2	Repräsentation ganzer Zahlen in Java – das Zweierkomplement	1135
22.1.3	Das binäre (Basis 2), oktale (Basis 8), hexadezimale (Basis 16) Stellenwertsystem	1137
22.1.4	Auswirkung der Typumwandlung auf die Bit-Muster	1138
22.1.5	Vorzeichenlos arbeiten	1140
22.1.6	Die Verschiebeoperatoren	1143
22.1.7	Ein Bit setzen, löschen, umdrehen und testen	1146
22.1.8	Bit-Methoden der Integer- und Long-Klasse	1146
22.2	Fließkomma-Arithmetik in Java	1148
22.2.1	Spezialwerte für Unendlich, Null, NaN	1148
22.2.2	Standardnotation und wissenschaftliche Notation bei Fließkommazahlen *	1152
22.2.3	Mantisse und Exponent *	1152
22.3	Die Eigenschaften der Klasse Math	1154
22.3.1	Attribute	1154
22.3.2	Absolutwerte und Vorzeichen	1154
22.3.3	Maximum/Minimum	1155
22.3.4	Runden von Werten	1156
22.3.5	Rest der ganzzahligen Division *	1158
22.3.6	Division mit Rundung Richtung negativ unendlich, alternativer Restwert *	1159
22.3.7	Multiply-Accumulate	1161
22.3.8	Wurzel- und Exponentialmethoden	1161
22.3.9	Der Logarithmus *	1162
22.3.10	Winkelmethode *	1163
22.3.11	Zufallszahlen	1165
22.4	Genauigkeit, Wertebereich eines Typs und Überlaufkontrolle *	1165
22.4.1	Der größte und der kleinste Wert	1165
22.4.2	Überlauf und alles ganz exakt	1166
22.4.3	Was bitte macht eine ulp?	1169

22.5	Zufallszahlen: Random, SecureRandom, SplittableRandom	1170
22.5.1	Die Klasse Random	1171
22.5.2	Random-Objekte mit dem Samen aufbauen	1171
22.5.3	Einzelne Zufallszahlen erzeugen	1172
22.5.4	Pseudo-Zufallszahlen in der Normalverteilung *	1172
22.5.5	Strom von Zufallszahlen generieren *	1173
22.5.6	Die Klasse SecureRandom *	1174
22.5.7	SplittableRandom *	1175
22.6	Große Zahlen *	1175
22.6.1	Die Klasse BigInteger	1176
22.6.2	Beispiel: ganz lange Fakultäten mit BigInteger	1183
22.6.3	Große Fließkommazahlen mit BigDecimal	1184
22.6.4	Mit MathContext komfortabel die Rechengenauigkeit setzen	1187
22.6.5	Noch schneller rechnen durch mutable Implementierungen	1188
22.7	Mathe bitte strikt *	1188
22.7.1	Strikte Fließkommaberechnungen mit strictfp	1189
22.7.2	Die Klassen Math und StrictMath	1189
22.8	Geld und Währung	1190
22.8.1	Geldbeträge repräsentieren	1190
22.8.2	ISO 4217	1190
22.8.3	Währungen in Java repräsentieren	1191
22.9	Zum Weiterlesen	1192

23 Testen mit JUnit 1193

23.1	Softwaretests	1193
23.1.1	Vorgehen beim Schreiben von Testfällen	1194
23.2	Das Test-Framework JUnit	1194
23.2.1	Test-Driven Development und Test-First	1195
23.2.2	Testen, implementieren, testen, implementieren, testen, freuen	1196
23.2.3	JUnit-Tests ausführen	1198
23.2.4	assertXXX(...)-Methoden der Klasse Assert	1198
23.2.5	Hamcrest	1201
23.2.6	Exceptions testen	1205
23.2.7	Tests ignorieren	1207
23.2.8	Grenzen für Ausführungszeiten festlegen	1207
23.2.9	Mit Methoden der Assumptions-Klasse Tests abbrechen	1208
23.3	Wie gutes Design das Testen ermöglicht	1208

23.4	Aufbau größerer Testfälle	1210
23.4.1	Fixtures	1211
23.4.2	Sammlungen von Testklassen und Klassenorganisation	1213
23.5	Dummy, Fake, Stub und Mock	1214
23.6	JUnit-Erweiterungen, Testzusätze	1215
23.7	Zum Weiterlesen	1216
24	Die Werkzeuge des JDK	1217
<hr/>		
24.1	Übersicht	1217
24.1.1	Aufbau und gemeinsame Schalter	1218
24.2	Java-Quellen übersetzen	1218
24.2.1	Java-Compiler vom JDK	1218
24.2.2	Alternative Compiler	1219
24.2.3	Native Compiler	1220
24.2.4	Java-Programme in ein natives ausführbares Programm einpacken	1220
24.3	Die Java-Laufzeitumgebung	1221
24.3.1	Schalter der JVM	1221
24.3.2	Der Unterschied zwischen java.exe und javaw.exe	1224
24.4	jlink: der Java Linker	1224
24.5	Dokumentationskommentare mit Javadoc	1225
24.5.1	Einen Dokumentationskommentar setzen	1226
24.5.2	Mit dem Werkzeug javadoc eine Dokumentation erstellen	1228
24.5.3	HTML-Tags in Dokumentationskommentaren *	1229
24.5.4	Generierte Dateien	1229
24.5.5	Dokumentationskommentare im Überblick *	1229
24.5.6	Javadoc und Doclets *	1231
24.5.7	Veraltete (deprecated) Typen und Eigenschaften	1231
24.5.8	Javadoc-Überprüfung mit DocLint	1234
24.6	Das Archivformat JAR	1235
24.6.1	Das Dienstprogramm jar benutzen	1236
24.6.2	Das Manifest	1238
24.6.3	Applikationen in JAR-Archiven starten	1239
24.7	Zum Weiterlesen	1240
Anhang	1241
Index	1261