

HANSER



Leseprobe

Fritz Jobst

Programmieren in Java

ISBN (Buch): 978-3-446-44134-7

ISBN (E-Book): 978-3-446-44150-7

Weitere Informationen oder Bestellungen unter

<http://www.hanser-fachbuch.de/978-3-446-44134-7>

sowie im Buchhandel.

Inhalt

Vorwort	XI
1 Der Einstieg in Java	1
1.1 Erstellung und Ablauf von Programmen in Java	2
1.2 Das erste Java-Programm	3
1.3 Erstellung und Ablauf des ersten Programms	5
1.4 Ein erstes Applet	9
1.5 Zusammenfassung	11
1.6 Aufgaben	12
2 Elemente der Programmierung	13
2.1 Daten erklären und verarbeiten	14
2.1.1 Schreibweisen für Deklarationen und Wertzuweisungen	16
2.1.2 Beispiel: Elementare Ausdrücke	18
2.1.3 Beispiel: Bereichsüberschreitungen	20
2.1.4 Typumwandlungen	21
2.1.5 Deklarationen mit dem static-Modifizierer	22
2.1.6 Namen und ihre Gültigkeit	23
2.2 Kontrollfluss	24
2.2.1 Verzweigung	25
2.2.2 Mehrfachverzweigung	29
2.2.3 Schleifen mit Vorabprüfung	32
2.2.4 Schleife mit Prüfung am Ende	36
2.2.5 Verlassen von Schleifen	38
2.2.6 Programmausnahmen	39
2.3 Methoden	43
2.3.1 Definitionen	43
2.3.2 Beispiele zum Einsatz von Methoden	45
2.3.3 Rekursion	48
2.3.3.1 Beispiel: Berechnung der Fakultät	49
2.3.3.2 Beispiel: Die Türme von Hanoi	50

2.4	Felder	54
2.4.1	Eindimensionale Felder	55
2.4.1.1	Grundlegende Definitionen	55
2.4.1.2	Beispiel: Einlesen und Bearbeiten eines Felds	58
2.4.1.3	Behandlung von Indexfehlern	59
2.4.2	Suche in Feldern	60
2.4.2.1	Lineare Suche	60
2.4.2.2	Halbierungsmethode: binäre Suche in Feldern	62
2.4.3	Sortieren	63
2.4.4	Mehrdimensionale Felder	66
2.5	Operatoren in Java	68
2.6	ANSI-Escape-Sequenzen	72
2.7	Aufgaben	73
3	Objektorientierte Programmierung	77
3.1	Auf dem Weg zu Klassen	77
3.1.1	Wege zur Objektorientierung	77
3.1.2	Beziehungen zwischen Klassen	80
3.1.3	Oberklassen und Unterklassen	81
3.1.4	Klassen und Objekte	82
3.1.5	Abstrakte Klassen	82
3.1.6	Entwurf der Klassen	83
3.1.6.1	Typ 1: die vorgegebenen Objekte	83
3.1.6.2	Typ 2: Verwaltungsobjekte oder Sammlungen	83
3.1.6.3	Typ 3: Umgebungsobjekte	83
3.1.6.4	Typ 4: übersehene Klassen doch noch finden	84
3.1.6.5	Gemeinsame Oberklassen finden	84
3.2	Klassen in Java	84
3.2.1	Eine Klasse zum Verwalten von Mitarbeitern	86
3.2.2	Erzeugung von Objekten: Konstruktoren	89
3.2.3	Wertzuweisung und Übergabe als Parameter	90
3.2.4	Statische Klassenelemente	91
3.2.4.1	Grundlagen	92
3.2.4.2	Initialisierung der static-Variablen in einer Klasse	93
3.2.5	Eingeschachtelte Klassen, innere Klassen	94
3.2.6	Umwicklertypen	96
3.3	Unterklassen und Polymorphie in Java	98
3.3.1	Definition von Unterklassen in Java	98
3.3.2	Methoden der Klasse Object überschreiben	101
3.3.3	Lebenszyklus von Objekten	102
3.3.4	Wie funktioniert die Polymorphie?	104
3.3.5	Wertzuweisung und Cast-Anweisung	105
3.3.6	Klassen und Ausnahmen	107
3.3.7	Abstrakte Klassen: Design für Vererbung	109

3.4	Generische Elemente in Java	113
3.5	Schnittstellen in Java	116
3.5.1	Vergleich von Objekten	118
3.5.2	Statische Methoden in Schnittstellen	121
3.5.3	Default-Methoden in Schnittstellen	121
3.5.4	Schnittstellen ohne abstrakte Methoden	123
3.5.5	Funktionale Schnittstellen	125
3.6	Funktionen in Java 8	126
3.6.1	Referenzen auf Funktionen in Java 8	126
3.6.2	Lambda-Ausdrücke	127
3.6.2.1	Syntax für Lambda-Ausdrücke	127
3.6.2.2	Bindung in Lambda-Ausdrücken	128
3.6.3	Beispiel: Anwendung der Comparator-Schnittstelle	129
3.6.4	Beispiel: Funktionen als Parameter	130
3.6.5	Zusammenfassung	132
3.7	Dynamische Erzeugung von Objekten	132
3.8	Aufzählung von Konstanten mit enum	137
3.9	Allgemeine Eigenschaften	140
3.9.1	Der final-Modifizierer	140
3.9.2	Packages mit package, import	140
3.9.3	Sichtbarkeit von Namen in Java	142
3.9.4	Wiederherstellung des Zustands eines Objekts: Serialisierung	142
3.9.5	Zusicherungen	144
3.10	Aufgaben	146
4	Grundlegende Klassen	151
4.1	Nützliche Klassen und Packages	151
4.1.1	Übersicht der Aufgaben des Package java.lang	152
4.1.2	Zeichenketten in Java: String	152
4.1.3	Die Klasse System	156
4.1.4	Die Klasse Math	156
4.1.5	Zeit und Datum in Java	157
4.1.6	Reflexion von Java-Programmen	160
4.1.7	Annotationen	162
4.1.8	Reguläre Ausdrücke	164
4.1.9	Protokollierung von Programmläufen: Logging	167
4.2	Verwalten von Objekten mit Sammlungen	169
4.2.1	Prinzip für die Aufbewahrung von Objekten	170
4.2.1.1	Schnittstellen für die Sammlungen in Java	171
4.2.1.2	Implementierungen für die Schnittstellen	172
4.2.2	Sequenzieller Zugriff: List, Set und SortedSet	173
4.2.2.1	Collection als Basisschnittstelle	173
4.2.2.2	Listen	175
4.2.2.3	Die ListIterator-Schnittstelle	178
4.2.2.4	Mengen	180

4.2.3	Assoziativer Zugriff: Map	184
4.2.3.1	Map als Basisschnittstelle	185
4.2.3.2	Die SortedMap-Schnittstelle	188
4.2.4	Nützliche Klassen und Methoden für Sammlungen	189
4.2.4.1	Die Klasse Collections	189
4.2.4.2	Implementierungen von Sammlungen für spezielle Zwecke ...	190
4.2.4.3	Sammlungen und Threads	191
4.2.4.4	Nützliche Klassen und Methoden	191
4.3	Streams in Java	192
4.3.1	Einstieg in die funktionale Programmierung	193
4.3.2	Ausgewählte Methoden	195
4.3.3	Fallbeispiele – Anwendungsfälle	197
4.3.3.1	Zeichenketten aneinanderhängen	197
4.3.3.2	Sortieren	198
4.3.3.3	Gruppieren von Objekten nach diversen Kriterien	199
4.3.3.4	Verarbeiten von Daten in Textdateien	200
4.3.3.5	Berechnungen mit Zwischenergebnissen durchführen	201
4.4	Aufgaben	202
5	Ein-/Ausgabe in Java	205
5.1	Prinzip der Ein-/Ausgabe in Java	208
5.1.1	Eingabe in Java	210
5.1.1.1	InputStream als Basisklasse für Eingaben	210
5.1.1.2	Reader als Brücke zwischen Bytes und Zeichen	212
5.1.2	Ausgabe in Java	214
5.1.2.1	OutputStream als Basisklasse für Ausgaben	214
5.1.2.2	Die Writer-Klassen in Java	219
5.2	Fallstudien zu Ein-/Ausgabe	220
5.2.1	Bearbeiten von Textdateien	220
5.2.2	Durchlaufen aller Dateien in einem Verzeichnis	222
5.2.3	Zugriff auf die Einträge in einem ZIP-Archiv	223
5.3	Aufgaben	224
6	Nebenläufigkeit in Java: Threads	227
6.1	Einstieg in Threads in Java	227
6.1.1	Implizite Programmierung von Threads	228
6.1.2	Streams parallel bearbeiten	229
6.1.3	Explizite Programmierung von Threads	230
6.2	Grundlagen zu Threads	232
6.2.1	Nutzen von Threads	232
6.2.2	Wettrennen	235
6.2.3	Zustände von Threads	239
6.2.4	Wichtige Methoden für Threads	240

6.3	Monitore in Java	241
6.3.1	Grundlagen des Monitorkonzepts in Java	241
6.3.2	Anwendung der Monitore in Java	241
6.4	Anwendungsfälle	243
6.4.1	Lang laufende Aktivitäten in Benutzungsoberflächen	244
6.4.2	Erzeuger-Verbraucher-Kopplung	247
6.4.3	Leser-Schreiber-Problem	249
6.4.4	Semaphoren	249
6.4.5	Verklemmungen und die fünf Philosophen	251
6.4.6	Animationen	253
6.5	Aufgaben	256
7	Grafikanwendungen in Java	259
7.1	Struktur von GUI-Anwendungen	259
7.1.1	Ein erstes Programm für Swing	260
7.1.2	Prinzip der ereignisgesteuerten Programmierung	261
7.1.3	Ereignissteuerung	262
7.1.3.1	Das „Delegation Event Model“	264
7.1.3.2	Listener-Schnittstellen und Adapter	266
7.1.4	Hierarchie der Swing-Klassen für Steuerelemente	267
7.1.5	Elementare Steuerelemente	268
7.1.6	Das Model-View-Controller-Paradigma und Swing	269
7.2	Anordnung der Komponenten	269
7.2.1	BorderLayout	271
7.2.2	FlowLayout	271
7.2.3	GridLayout	272
7.2.4	CardLayout	273
7.2.5	GridBagLayout	274
7.2.6	BoxLayout (nur Swing)	277
7.2.7	Schachtelung der Layouts	278
7.3	Steuerelemente in Benutzeroberflächen	279
7.3.1	Schaltflächen: JButton	279
7.3.2	Checkboxes und Radiobutton	280
7.3.3	Statischer Text zur Anzeige von Informationen	281
7.3.4	Listen zur Auswahl	282
7.3.5	Elementare Auswahl mit der JComboBox	284
7.3.6	Textfelder	285
7.3.7	Menüs in Java	286
7.4	Steuerelemente unter der MVC-Architektur	288
7.4.1	Übersicht: Aufgabenverteilung Swing-Anwender	289
7.4.2	Vertiefung für JList und JComboBox	290
7.4.2.1	Eine MVC-Anwendung für JList	291
7.4.2.2	JComboBox	293

7.4.3	Tabellen und Baumsteuerelemente	294
7.4.3.1	Das Steuerelement für Tabellen JTable	294
7.4.3.2	JTree	298
7.5	Kurs: GUI-Anwendungen	304
7.5.1	Erstellung einer grafischen Komponente	305
7.5.2	Reaktion auf Mausklicks	307
7.5.3	Reaktion auf Mausbewegungen: ein Malprogramm	309
7.5.4	Turtle-Grafik	311
7.5.5	Dialoge in Java	315
7.5.6	Dialog zur Auswahl von Dateinamen	319
7.5.7	Die Türme von Hanoi	321
7.6	Aufgaben	325
8	Programmierung in Netzwerken	331
8.1	Programmierung von Sockets	331
8.1.1	Grundlagen von Netzwerken	332
8.1.2	Sockets in Java	333
8.1.2.1	Verbindungsorientierte Kommunikation mit TCP	333
8.1.2.2	Verbindungslose Kommunikation	336
8.1.2.3	Java-Klassen für Internetadressen und zur Darstellung von URLs	337
8.2	Verteilte Anwendungen	338
8.2.1	Der Additions-Dienst mit TCP	339
8.2.1.1	Problemanalyse: Datenaustausch	339
8.2.1.2	Problemanalyse: Aufbau der Anwendung	340
8.2.2	Beispiel: der Additions-Dienst mit UDP	342
8.2.2.1	Problemanalyse: Datenaustausch	342
8.2.2.2	Problemanalyse: Aufbau der Anwendung	342
8.2.3	RMI	344
8.2.3.1	Problemanalyse: Datenaustausch	344
8.2.3.2	Problemanalyse: Aufbau der Anwendung	345
8.2.4	Prinzip von RMI	345
8.3	Aufgaben	348
9	Anbindung von Datenbanken mit JDBC	349
9.1	Grundlagen von JDBC	350
9.1.1	Grundsätzlicher Ablauf beim Zugriff	350
9.1.2	Einstieg in relationale Datenbanken und SQL	353
9.1.3	Klassen und Schnittstellen im Package java.sql	355
9.2	Beispiel: Die Datenbank mit den Speisen in JDBC	362
9.2.1	Programmierung der Verbindung zu den Datenbanken	362
9.2.2	Vorbereitung: Datenbanken einrichten	363
9.2.3	Zugriffe mit JDBC	364
9.2.3.1	Die Speisedatenbank	364

9.2.3.2	Die Speisedatenbank: objektrelationale Zuordnung	364
9.2.3.3	Die Speisedatenbank: Vermeidung doppelter Einträge	365
9.3	Datentypen in Java und SQL	367
9.4	Metadaten	368
9.4.1	Metadaten und die Auskunft über die Datenbank	368
9.4.2	Anwendung	369
9.5	Aufgaben	371
10	Bearbeiten von XML in Java	373
10.1	Schreiben von XML und Lesen mittels JAXB	374
10.1.1	Zusammenhänge: Klasse und Objekt bzw. Schema und XML	375
10.1.2	Kochrezept: Anleitung zur Benützung von JAXB	377
10.1.2.1	Schritt: Kennzeichnung eines Objekts zum Speichern	377
10.1.2.2	Schritt: Programm zum Speichern und Laden	377
10.2	SAX-Parser	378
10.3	Aufgaben	381
11	Werkzeuge für die Java-Programmierung	383
11.1	Der Compiler javac	383
11.2	Der Interpreter java	384
11.3	Applet-Viewer	384
11.4	Der Generator für die Dokumentation	385
11.5	Der Disassembler	385
Literatur	387
Index	389

Vorwort

Liebe Leserin, lieber Leser,

in der objektorientierten Sprache Java programmieren wir seit 1996 für diverse Systeme. Jetzt können wir mit Java 8 für manche Probleme einfachere und prägnantere Programme im funktionalen Stil formulieren.

Wir fokussieren auf zentrale Techniken der Programmierung und nutzen dabei den Komfort und die Leistungsfähigkeit der Entwicklungsumgebung Eclipse gleich beim Start ins Java-Land in *Kapitel 1*. In *Kapitel 2* besprechen wir die Grundlagen der Steuerung von Abläufen in Programmen, der sog. prozeduralen Programmierung. Erst wenn wir Verzweigungen, Schleifen und Aufrufe von Funktionen beherrschen, können wir uns in *Kapitel 3* den Herausforderungen der Objektorientierung stellen. Dort lernen wir auch die Lambda-Ausdrücke als neue Möglichkeiten in Java 8 zur Implementierung funktionaler Schnittstellen kennen.

Ein einzelnes Objekt kommt selten alleine. Zur Verwaltung von Objekten dienen die sog. Collections. Praktiker wissen, dass man mit den sequentiellen und assoziativen Sammlungen die meisten alltäglichen Probleme der Verwaltung von Daten lösen kann, auf die wir uns im Buch konzentrieren. Java unterstützt uns bei der Programmierarbeit auch mit anderen Klassen, von denen wir in *Kapitel 4* eine kleine, praxisorientierte Auswahl vorstellen. Mit Java 8 können wir mit Streams die Iteration über diese Datenbestände dem Laufzeitsystem überlassen, wir spezifizieren nur, was getan werden soll.

Kapitel 5 führt in die Grundlagen der Verarbeitung von Daten auf externen Speichermedien ein. Dazu nutzen wir die neuen Möglichkeiten von Java 8 zur vereinfachten Programmierung. Zum Verständnis der Programmierung von parallelen Streams und grafischen Benutzungsoberflächen machen wir uns in *Kapitel 6* mit den Chancen und Risiken der nebenläufigen Programmierung vertraut.

In *Kapitel 7* führen wir die Grundkonzepte der Programmierung grafischer Benutzungsoberflächen ein. Wir lernen die Steuerelemente sowie die Möglichkeiten des Aufbaus kennen. Die Anbindung von Programmcode an Benutzungsoberflächen gelingt uns mit den Lambda-Ausdrücken von Java 8 kürzer und prägnanter als bisher.

Java ist als Programmiersprache für Anwendungen im Internet konzipiert. Deswegen können wir in *Kapitel 8* mit relativ geringem Aufwand Programme für verteilte Anwendungen erstellen.

In *Kapitel 9* lernen wir den Zugriff auf Datenbanken. Damit können wir unsere Objekte in Datenbanken ablegen und daraus wiedergewinnen. Mit den Basistechniken zur Speicherung bzw. des Lesens von Daten im XML-Format in *Kapitel 10* runden wir unseren Streifzug durch das Java-Land ab.

Wäre es nicht einfach schön, das eigene Android-Smartphone programmieren zu können? Es gibt zwar viele Apps, denn Android gewinnt immer höhere Anteile am Markt für Smartphones. Dennoch fasziniert es, Anwendungen für das eigene Gerät selbst zu schreiben. Mit dem Buch „Programmieren in Java“ erhalten Sie eine solide Grundlage in der Programmiersprache Java. Damit könnten Sie Apps für Android selbst entwickeln. Aber aller Anfang ist schwer, so auch der Start in Android. Deswegen bietet dieses Buch Ergänzungen zum Download als Hilfe zum Einstieg in die Entwicklung für Android an.

Kapitel 1 der Ergänzungen zu diesem Buch begleitet Sie auf dem Weg der Installation des Entwicklungssystems für Android. Damit schaffen wir die erste App für Android.

Im Gegensatz zum PC benötigen wir am Smartphone bereits für einfachste Anwendungen Grundkenntnisse im Umgang mit grafischen Benutzungsoberflächen. Kapitel 7 dieser Ergänzungen führt Sie in die Grundproblematik ein. Dabei müssen wir auch einige Besonderheiten bei Android-Smartphones beachten.

Meine Empfehlung zum Einstieg in die Programmierung für Android: Arbeiten Sie die Kapitel 1, 2, 3 und 4 des Buchs „Programmieren in Java“ durch. Danach können Sie über Kapitel 1 der Ergänzungen zu diesem Buch in die Entwicklung für Android in Kapitel 7 der Ergänzungen einsteigen.

Wie bei Fremdsprachen gilt auch bei Programmiersprachen: Man muss die Sprache sprechen, d. h. selbst programmieren. Deswegen enthält jedes Kapitel kleinere oder größere Aufgaben. Das Spektrum der Aufgaben reicht von elementaren Übungen bis zu kleinen Projektarbeiten. Die Ergänzungen für Android enthalten die analogen Aufgabenstellungen für Android-Apps. Lösungsvorschläge zu allen Aufgaben finden Sie über das Internetportal des Carl Hanser Verlags <http://downloads.hanser.de>. Von dort können Sie auch alle Programme im Buch in vollständiger Form herunterladen.

Mein besonderer Dank gilt Frau Brigitte Bauer-Schiewek für die aufmerksame Begleitung bei der Konzeption und Durchführung sowie Frau Irene Weilhart für ihre Präzision und Sorgfalt bei der Herstellung.

Viel Erfolg mit diesem Buch!

Regensburg, im November 2014

Fritz Jobst

4.3.3 Fallbeispiele – Anwendungsfälle

In diesem Abschnitt stellen wir mögliche Anwendungsfälle für Streams vor. Besonders nützlich zum Aufsammeln von Ergebnissen sind die Methoden der Klasse `Collectors` aus dem API von Java 8. Sie bietet nützliche Operationen zur Reduktion, wie das Aufsummieren der Elemente, das Zusammenfassen aller Ergebnisse in einer geeigneten `Collection` oder das Gruppieren nach bestimmten Kriterien. Hiermit lösen wir Standardprobleme wie:

- Zeichenketten aneinanderhängen
- Sortieren
- Gruppieren von Objekten nach diversen Kriterien
- Verarbeiten von Textdateien
- Berechnungen durchführen

4.3.3.1 Zeichenketten aneinanderhängen

Mit Hilfe der `joining`-Methoden der Klasse `java.util.stream.Collectors` in Tabelle 4.29 können wir Zeichenketten bequem zusammenhängen. In der ersten Variante fügen wir zwischen zwei Texten ein Komma ein, in der zweiten Variante fügen wir zusätzlich vor und nach dem Ergebnis noch Zeichenketten ein. In Variante 3 benützen wir Methoden der `StringBuilder`-Klasse, um den Prozess des Aufsammelns der Zeichenketten zu kontrollieren. Dazu müssen wir drei Referenzen zu Methoden angeben:

1. Einrichten eines Suppliers. Er richtet ein Objekt zum Aufsammeln ein.
2. Methode zum Anhängen eines Elements an ein Objekt zum Aufsammeln
3. Methode zum Aneinanderhängen zweier Objekte zum Aufsammeln

```
List<String> strings = Arrays.asList("a", "b", "c", "d");
String c1 = strings
    .stream()
    .collect(Collectors.joining(","));
System.out.println(c1);    // Ergebnis = a,b,c,d

String c2 = strings
    .stream()
    .collect(Collectors.joining(", ", "[", "]"));
System.out.println(c2);    // Ergebnis = [a, b, c, d]

String c3 = strings
    .stream()
    .collect(
        StringBuilder::new,           // a) Einrichten
        StringBuilder::append,       // b) Anhängen eines Strings
        StringBuilder::append)       // c) Zusammenhängen zweier StringBuilders
    .toString();
System.out.println(c3);    // Ergebnis = abcd
```

4.3.3.2 Sortieren

Zum Sortieren von Elementen, die die `Comparable`-Schnittstelle implementieren, dient die `sorted()`-Methode für Streams. Im folgenden Beispiel übergeben wir einen `Comparator` als Lambda-Ausdruck, um in absteigender Reihenfolge zu sortieren.

```
List<String> strings = Arrays.asList("a", "b", "c", "d");
strings.stream()
    .sorted((a,b) -> b.compareTo(a))
    .forEach(s->System.out.printf("%s", s));
```

Sortierte Ausgabe aller Einträge in einem Verzeichnis

Wenn wir alle Einträge vom Typ `File` eines Verzeichnisses ausgeben wollen, lesen wir die Katalogeinträge zunächst mit Hilfsmitteln aus Kapitel 5 ein (siehe auch Abschnitt 5.2.4). Als Ergebnis erhalten wir alle Einträge in einer Liste. Jetzt können wir die Einträge nach verschiedenen Kriterien sortieren. Weil die `Comparator`-Schnittstelle eine funktionale Schnittstelle ist, können wir den Vergleich mit einem Lambda-Ausdruck formulieren. Die Liste der Dateien liefert einen Stream, für den wir mit der `sorted(...)`-Methode einen je nach Kriterium sortierten Stream erhalten. Die Elemente geben wir dann aus.

Listing 4.24 Beispiel: Dateien eines Verzeichnisses nach diversen Kriterien sortiert ausgeben

```
public class DirectoryLister {
    private static final Comparator<File> NAMEORDER
        = (f1, f2) -> f1.getName().compareTo(f2.getName());

    private static final Comparator<File> SIZEORDER
        = (f1, f2) -> Long.valueOf(f1.length()).compareTo(f2.length());

    public DirectoryLister(String dirName) {
        File dir = new File(dirName);
        String fileNames[] = dir.list();
        List<File> files = new ArrayList<>(fileNames.length);

        for (String fileName : fileNames) {
            File temp = new File(dir, fileName);
            files.add(temp);
        }

        // Ausgabe nach Groesse geordnet (groesste Datei zuletzt)
        files.stream().sorted(SIZEORDER).forEach(System.out::println);
        System.out.println();
        // Ausgabe nach Name geordnet
        files.stream().sorted(NAMEORDER).forEach(System.out::println);
        System.out.println ();
    }

    public static void main (String[] args) {
        new DirectoryLister (".\src");
    }
}
```

4.3.3 Gruppieren von Objekten nach diversen Kriterien

Zum Gruppieren von Objekten benötigt man ein Kriterium. Als Ergebnis erhält man für jeden Wert des Kriteriums eine u.U. leere Sammlung von Objekten, die diesem Kriterium genügen.

Gruppieren von Zahlen

Im ersten Beispiel wollen wir eine Folge von Zahlen in ungerade und gerade Zahlen gruppieren. Zum Test dividieren wir eine Zahl durch 2. Wenn der Rest bei der Division gleich 0 ist, haben wir eine gerade Zahl, ansonsten eine ungerade Zahl. Ein Lambda-Ausdruck liefert das Ergebnis, welches wir an die Methode `Collectors.groupingBy` übergeben. Als Ergebnis erhalten wir eine Map zuordnung von Boolean zu Listen von Zahlen. Wir starten eine Iteration über die Schlüssel und geben den Wert jedes Schlüssels sowie die ihm zugeordneten Zahlen aus.

Listing 4.25 Gruppieren von Essen-Objekten nach Typ

```
Map<Boolean, List<Integer>> zuordnung =
    Stream.of(1,2,3,4,5)
        .collect(Collectors.groupingBy(i -> i %2 == 0));

zuordnung.keySet() // Menge der Schlüssel auslesen
    .stream() // Stream aus Schlüssel bilden
    .forEach(k -> { // Für jedes Paar (Schlüssel Liste...)
        System.out.printf ("\n%s:", k.toString()); // Schlüssel ausgeben
        li.get(k) // Liste holen
            .stream() // Stream der einzelnen Zahlen bearbeiten
            .forEach(i -> System.out.printf ("%d,", i));
    });
```

Ergebnis

```
false:1,3,5,
true:2,4,
```

Gruppieren von Objekten

Im zweiten Beispiel wollen wir Objekte mit einem enum-Attribut nach diesem Attribut gruppieren. Als Objekte nehmen wir die Essen-Objekte aus Kapitel 7.

```
public enum EssenTyp {
    OBST, GEMUESE, FASTFOOD, SLOWFOOD, ETC;
}

public class Essen {
    private String name;
    private String kommentar;
    private EssenTyp essenTyp; // Kriterium zum Gruppieren: Alle GEMUESE etc...
    //.. Dazu Konstruktoren, get- und set-Methoden, siehe Kapitel 7
    // z.B. public EssenTyp getEssenTyp () { return essenTyp; }
}

Stream<Essen> essen = Arrays.stream(Essen.getAlleEssen());
```

```

Map <EssenTyp, List<Essen>> mapEssen =
    essen.collect(Collectors.groupingBy(e -> e.getEssenTyp()));
// Für alle Schlüssel tue
// Schlüssel ausgeben, alle Werte zum Schlüssel ausgeben
mapEssen.keySet()
    .stream()
    .forEach(key -> {
        System.out.printf("\nTyp = %s\n", key);
        mapEssen.get(key)
            .stream()
            .forEach(e -> System.out.printf(" Essen = %s\n", e));
    });

```

Ergebnis (abgekürzt)

```

Typ = GEMUESE
  Essen = Broccoli
  Essen = Karotte
  Essen = Mais
  Essen = Rettich

Typ = FASTFOOD
  Essen = Hamburger
  Essen = Pommes usw...

```

4.3.3.4 Verarbeiten von Daten in Textdateien

Wir wollen alle Worte einer Textdatei bearbeiten, z.B. alle Zahlen aufsummieren. Eine Textdatei besteht aus einzelnen Zeilen. Diese Folge von Zeilen können wir mit `java.nio.file.Files.lines` als Stream lesen. Jede dieser Zeilen enthält ihrerseits eine Folge von Worten. Wir haben es hier mit einem Stream (aus Zeilen) zu tun, wobei jedes Element des Streams seinerseits wieder aus Streams (aus Worten) besteht. Solche „Streams aus Streams“ mit einer Art 1:n-Beziehung können wir mit der `flatMap`-Operation zu einem Stream (aus Worten) zusammenhängen und diesen Stream durchlaufen. In Listing 4.26 wandeln wir dies in einen `IntStream` um, den wir mit `sum()` aufaddieren können.

Listing 4.26 Addieren aller Zahlen einer Textdatei mit ganzen Zahlen

```

public class ZahlenAddieren {

    public static void main(String[] args) {
        String dateiName = "zahlen.txt";
        try {
            System.out.println (
                Files.lines(Paths.get(dateiName))           // Stream aus Zeilen
                    .flatMap(line->Stream.of(line.split(" +"))) // Stream aus allen Texten
                    .mapToInt(s -> Integer.parseInt(s))      // IntStream
                    .sum ());                                 // Summe berechnen
        } catch (IOException e) {
            e.printStackTrace(System.err);
        }
    }

}

```

4.3.3.5 Berechnungen mit Zwischenergebnissen durchführen

Für die Berechnung von Summen aus `int` ganzen Zahlen bietet die Klasse `IntStream` die Methode `sum()`, für Durchschnitte die Methode `average()`. Will man Summen aus Integer-Objekten berechnen, steht keine der o.a. Methoden zur Verfügung. Für Summen kann man die `reduce(...)`-Anweisung benutzen:

```
int summe = Stream.of (3, 5, usw... 3 ).reduce(0, (l, r) -> l + r);
```

Dies hilft bei der Berechnung des Durchschnitts nicht weiter, denn zur Berechnung des Durchschnitts benötigt man auch die Anzahl der Elemente. Diese könnte man separat mit der `count()`-Methode bestimmen. Wenn man die Folge der Elemente nicht zweimal durchlaufen will, reicht eine Schnittstelle nicht aus, wir benötigen eine Klasse, da die Attribute der Klasse Daten aufnehmen können. Dieses Verfahren setzen wir immer dann ein, wenn wir bei der Verarbeitung Daten benötigen, z. B. in Form übergebener Parameter.

Wir verwenden in Analogie zu einem Beispiel aus dem Java-API [API14] eine Klasse `Summierer`, die sowohl Summen bildet, als auch die Anzahl der Elemente mitzählt. Damit können wir in dieser Klasse eine Methode zur Berechnung des Durchschnitts angeben.

Wir berechnen den Durchschnitt in einer `Optional`-Klasse, da wir für den Fall einer leeren Sammlung von Elementen eine Division durch 0 vermeiden wollen. Siehe hierzu auch Aufgabe 4.3.

Listing 4.27 Berechnung von Durchschnittswerten

```
public class Zwischenergebnisse {
    public static class Summierer implements Consumer<Integer> {

        private int total = 0;
        private int count = 0;

        public Optional<Double> average() {
            return Optional.ofNullable(count > 0 ? ((double) total) / count : null);
        }

        @Override
        public void accept(Integer i) {
            total += i;
            count++;
        }

        public void combine(Summierer other) {
            total += other.total;
            count += other.count;
        }
    }

    public static void main(String[] args) {
        Optional<Double> d = Stream.of (3, 5, usw... 3 )
            .collect(Summierer::new,
                Summierer::accept,
                Summierer::combine)
            .average();
        System.out.println (d);
    }
}
```

Zusammenfassung

Java stellt im Package `java.util` Klassen zur Verfügung, die dem Programmierer die Entwicklung eigener Routinen für immer wieder zu lösende Probleme ersparen. Das `Collection`-Framework stellt ein leistungsfähiges System zur Aufbewahrung von Objekten zur Verfügung. Insbesondere kann man sequenziell anfallende Daten mit den Implementierungen der `List`-Schnittstelle verwalten. Wünscht man Sammlungen, die frei von Duplikaten sind, so kann man mit den Implementierungen der `Set`-Schnittstelle arbeiten. Für assoziative Zugriffe auf Daten bieten sich die Implementierungen der `Map`-Schnittstelle an.

Wir können alle Elemente von Sammlungen mit Hilfe der `Iterator`-Schnittstelle durchlaufen. Die erweiterte Form der `for`-Schleife vereinfacht dies in vielen Fällen. Ab Java 8 können wir mit Streams Grundaufgaben der Programmierung im funktionalen Stil effizient lösen: Sortieren, Gruppieren, elementare Operationen wie Saldieren usw.

■ 4.4 Aufgaben

Aufgabe 4.1

Benutzen Sie die Klasse zur Verwaltung von Mitarbeitern aus Abschnitt 3.2.1. Verwalten Sie Personen mit einer generischen Liste. Implementieren Sie Methoden zum Hinzufügen von Personen und zum Ausgeben aller Personen.

Aufgabe 4.2

Benutzen Sie die Klasse zur Verwaltung von Personen aus Abschnitt 3.2.1. Verwalten Sie die Personen mit einer `HashMap`. Implementieren Sie Methoden zum Hinzufügen von Personen und zum Suchen der Personen durch Angabe des Namens.

Aufgabe 4.3: zu `IntStream`

In dieser Aufgabe gehen wir von ganzen Zahlen im `int[]`-Array aus. Bitte bearbeiten Sie die Aufgabe mit `IntStream`, nicht mit `Stream<Integer>` wie in der nächsten Aufgabe.

```
int[] zahlen = {3, 5, 1, 3, 7, 29, 33, 49, 5, 1, 1, 2, 3};
```

1. Berechnen Sie die Summe aller Zahlen.
2. Berechnen Sie den Durchschnitt aller Zahlen.
3. Bestimmen Sie das Maximum aller Zahlen.
4. Geben Sie ein Histogramm der Zahlen an. Gruppieren Sie dazu die Zahlen nach Werten und ermitteln Sie für jeden Wert die Anzahl der Zahlen mit diesem Wert. Siehe Abschnitt 4.3.3.3.
5. Geben Sie die Zahlen in der Form `[3, 5, 3, ... 1, 1, 2, 3]` aus. Siehe Abschnitt 4.3.3.1.

Aufgabe 4.4 zu Stream<Integer>

In dieser Aufgabe gehen wir von Objekten in einem Array `Integer[]` aus und arbeiten mit `Stream<Integer>`.

```
Integer[] zahlen = { 3, 5, 1, 3, 7, 29, 33, 49, 5, 1, 1, 2, 3 };
```

1. Berechnen Sie die Summe aller Zahlen.
2. Berechnen Sie den Durchschnitt aller Zahlen.
3. Bestimmen Sie das Maximum aller Zahlen.
4. Geben Sie ein Histogramm der Zahlen an. Gruppieren Sie dazu die Zahlen nach Werten und ermitteln Sie für jeden Wert die Anzahl der Zahlen mit diesem Wert. Siehe Abschnitt 4.3.3.3.
5. Geben Sie die Zahlen in der Form `[3, 5, 3, ... 1, 1, 2, 3]` aus. Siehe Abschnitt 4.3.3.1.

Aufgabe 4.5: Primzahlen mit Streams berechnen

Gegeben ist die folgende recht ineffiziente Primzahl-Testmethode für Zahlen ≥ 2 . Diese Methode ist ein Prädikat für ganze Zahlen im Sinne von Java 8.

```
static boolean isPrime(int zahl) {  
    int teiler = 2;  
    while (zahl % teiler != 0)  
        teiler++;  
    if (zahl == teiler)  
        return true;  
    return false;  
}
```

1. Geben Sie die ersten 100 Primzahlen aus
2. Geben Sie die ersten 1000 Primzahlen aus. Nützen Sie alle Kerne Ihres Rechners zur Berechnung!
3. Aufgabe (optional)
Wenn Ihnen die obige Methode zur Ermittlung von Primzahlen zu primitiv ist, geben Sie eine effizientere Variante an.

Aufgabe 4.6

In Listing 4.25 aus Abschnitt 4.3.3.3 zeigen wir, wie man einzelne Essen nach dem Typ gruppiert. Ziel dieser Aufgabe ist es, die o. a. Objekte nach dem Anfangsbuchstaben ihres Namens zu gruppieren.

Aufgabe 4.7

Gegeben ist eine Datei mit Fußballern in der folgenden Form:

Listing 4.28 Die Nationalmannschaft im Herren-Fußball

```
Trikot;Vorname;Name;Geburtstag;Verein;Spiele;Tore;Rolle;
1;Manuel;Neuer;27.03.1986;FC Bayern München;52;0;TORWART;
15;Erik;Durm;12.05.1992;Borussia Dortmund;1;0;VERTEIDIGUNG;
7;Bastian;Schweinsteiger;01.08.1984;FC Bayern München;108;23;MITTELFELD;
11;Miroslav;Klose;09.06.1978;Lazio Rom;137;71;ANGRIFF;
```

Jede Zeile enthält die Daten für einen Fußballer. Die einzelnen Werte sind durch einen Strichpunkt getrennt. Für Fußballer gibt es die im o. a. Beispiel definierten Rollen und keine weiteren. Sie finden die komplette Datei mit allen Fußballern einer Mannschaft bei den Unterlagen zu diesem Buch.

1. Entwickeln Sie eine Klasse `Fussballer` mit allen acht in der ersten Zeile genannten Attributen. Definieren Sie für die Rolle eine geeignete `enum`-Klasse.
2. Geben Sie einen Konstruktor für diese Klasse aus einer Zeile an. (Hinweis: Die `split()`-Methode dient zum Aufteilen von Zeichenketten.)
3. Geben Sie eine Klasse `Mannschaft` an. Diese Klasse soll eine Liste aller Fußballer enthalten: `List<Fussballer> mannschaft`, die sog. Mannschaft.
4. Lesen Sie die Mannschaft mit Hilfe von b) und einer Methode der Klasse `Files` ein. (Hinweis: `Files.readAllLines(Paths.get("fussball.txt"), Charset.defaultCharset())` liefert eine Liste mit den Zeilen einer Textdatei).
5. Geben Sie die Namen der Spieler der Mannschaft aus.
6. Geben Sie die Namen der zehn Spieler mit den meisten Toren aus.
7. Geben Sie die Namen der Fußballspieler gruppiert nach Anfangsbuchstaben in der folgenden Form aus:

```
B: {Boateng}
D: {Durm,Draxler}
G: {Ginter,Großkreutz,Götze}
... usw.
W: {Weidenfeller}
Z: {Zieler}
```

8. Ermitteln Sie per Java-Programm, welcher Verein die meisten Spieler für die Nationalelf abgestellt hat.

Hinweise

Für die Teilaufgaben 7. und 8. empfiehlt sich eine Gruppierung nach Anfangsbuchstaben der Namen bzw. nach den Vereinen der Spieler.

Index

Symbole

@Override 99
@XmlElement 377

A

AbstractListModel 289
AbstractTableModel 289
acos() 156
ActionListener 265
Adapter 266
add() 177, 180
addAll() 177
AdjustmentListener 265
Annotation 162
append() 154
Applet 9, 110, 230
– Breite, Höhe 35
appletviewer 384
ArithmeticException 40
Array 55
arraycopy() 57
ArrayList 172, 173
asin() 156
assert 144
Assoziative Suche 184
at() 158
atan() 156
atan2() 156
AtomicInteger 230
AtomicLong 238
Attribut 85
Ausdruck
– Lambda 127
Ausführungskontext 233
Ausnahme 39, 107
Autoboxing 96

average() 193
AWT 259

B

Beobachter-Muster 263
Bereichs-Ansichten 183
Beziehung
– Benutzt-Beziehung 80
– Hat-Beziehung 80
– Ist-Beziehung 80
Bildschirm aktualisieren 307
BinaryOperator 125, 128, 196
binarySearch() 189
bitweise Verarbeitung 70
Black-Box-Test 62
boolean 14, 26, 69, 71
Boolean 96
BorderLayout 270, 271
BOTH 275
BoxLayout 270, 277
break 29, 38
Browser 232
BufferedInputStream 206, 212
BufferedOutputStream 206, 214
BufferedReader 206, 212
BufferedWriter 206
Button 268
byte 14
Byte 96
ByteArrayInputStream 212
ByteArrayOutputStream 214

C

CallableStatement 356, 359
 Canvas 268
 capacity() 154
 CardLayout 270, 273
 case 29
 catch 39
 ceil() 156
 char 14
 Character 96
 charAt() 152, 154, 155
 Checkbox 268, 280
 Choice 268
 class 86
 Class 132, 152
 ClassCastException 106
 CLASSPATH 351, 383
 clear() 187
 Client-Server 331
 Cloneable 123
 close 207
 close() 211, 215
 collect(...) 195, 199
 Collections 169, 171, 172, 173
 commit() 356
 Comparable 118, 189
 Comparator 120, 128, 129, 196, 198
 compareTo() 152
 compareToIgnoreCase() 153
 Compiler 3
 Component 268
 ComponentListener 265
 Consumer 125
 Container 259, 267, 270
 ContainerListener 265
 contains() 180
 containsKey() 186
 containsValue() 186
 continue 38
 Controller 269, 289
 copy() 189
 Co-Routinen 232
 cos() 156
 count() 195
 CREATE (SQL) 353
 currentThread() 240
 currentTimeMillis() 156

D

DatabaseMetaData 361
 DataInputStream 212

DataOutputStream 214
 Datentypen 82
 DayOfWeek 157, 159
 DecimalFormat 218
 default 29
 DefaultComboBoxModel 289
 DefaultHandler 378
 Default-Konstruktor 102
 DefaultMutableTreeNode 299
 Dekrementieren 70
 Delegate 269
 delete() 155
 Dialog 315
 do 36
 doInBackground() 244
 DOM 374
 done() 244
 double 14
 Double 96
 DoubleFunction 130
 DoubleStream 193
 drawLine 35
 DriverManager 351
 DROP (SQL) 353
 DRY-Prinzip 78
 Duration 159

E

Eclipse

- Erstellen einer Anwendung 6
- Erstellung eines Applets 10
- Erstellung von Klassen 141
- Generieren von equals() und hashCode() 124
- Getter und Setter erzeugen 89
- .jar-Archiv in Build-Path 351
- Konstruktoren erzeugen 89
- Optimieren von imports 141

 Eingabe

- von Standard 213
- von Tastatur 213

 else 25
 endsWith() 153
 entrySet() 187
 enum-Konstanten 137
 EnumSet 138, 190
 equals() 91, 124, 153, 186
 equalsIgnoreCase() 153
 ereignisgesteuerte Programmierung 262
 Erzeuger 243
 Escape-Sequenzen 72
 execute() 364

executeUpdate() 364
 Exemple 82
 exit(...) 156
 exp() 156
 extends 99

F

Fakultät 33, 49
 Felder 54, 55
 - binäre Suche 62
 - Lineare Suche 60
 - mehrdimensional 66
 - Suche 60
 - Übergabe 48
 FileInputStream 206, 212
 FileOutputStream 206, 214
 FileReader 206, 212
 FileWriter 206, 219
 fill() 189
 filter() 189, 194, 195
 FilterInputStream 206, 211
 FilterOutputStream 206
 FilterReader 206
 FilterWriter 206
 final 86, 140
 finalize() 104
 finally 39, 109
 find() 165
 findFirst() 195
 first() 183
 firstKey() 189
 Fläche füllen 305
 flatMap(...) 195, 200
 float 14
 Float 96
 floor() 156
 FlowLayout 270, 271
 FocusListener 265
 for 32
 forEach() 195, 198
 format() 158
 forName() 132
 for-Schleife
 - erweitert 35
 - klassisch 32
 Frame 260
 from() 158
 Function 125, 196
 Fünf Philosophen 251

G

Garbage-Collection 104
 gc() 156
 get() 158, 177, 186
 getBytes() 153
 getChildAt(int) 300
 getChildCount() 300
 getColumnCount() 295
 getDepth() 301
 getElementAt(int i) 289, 290
 getFirstChild() 300
 getInetAddress() 335
 getInputStream() 335
 getLastChild() 300
 getLevel() 301
 getListCellRendererComponent() 291
 getLocalAddress() 335
 getObject() 367
 getOutputStream() 335
 getParent() 300
 getProperty() 156
 getResultSet() 357
 getRowCount() 295
 getSiblingCount() 300
 getValueAt(...) 295
 getWidth() 10
 GridBagLayout 270, 274
 gridheight 275
 GridLayout 270, 272
 gridx 274
 gridy 274
 groupingBy(...) 196, 199
 Grundrechenarten 19

H

hashCode() 124
 HashMap 172, 184
 HashSet 172, 180
 Hashtable 185
 hasNext() 179
 headMap() 188
 headSet() 183
 Hello World 4
 HORIZONTAL 275

I

if 25
 implements 117, 132
 import 140
 import static 23

indexOf() 153, 177
 Indexunterlauf 59
 InflaterInputStream 212
 InflaterOutputStream 214
 Inkrementieren 70
 InputStream 205, 206, 210
 InputStreamReader 206, 212
 insert() 155
 Instant 159
 Instanz 89
 int 14
 Integer 96
 interface 132
 Intervall 183
 IntStream 193
 Invariante 145
 is() 158
 isEmpty() 180, 186
 isLeaf() 301
 ItemListener 265
 Iterator 169, 178, 180

J

JApplet 260
 java 384
 javac 384
 javadoc 385
 javap 385
 JAXB 374
 JAXBContext 377
 JAXP 378
 JButton 268, 279
 JCheckBox 268
 JComboBox 268, 284, 288
 JComponent 267, 268
 JDBC 349
 JFrame 260
 JLabel 268, 281
 JList 268, 282, 288
 JMenuBar 268
 joining() 196, 197
 JPanel 268
 JProgressBar 245, 268
 JRadioButton 268, 280
 JScrollbar 268
 JScrollPane 282, 285
 JSplitPane 303
 JTable 268, 288, 294
 JTextArea 268, 285
 JTextComponent 268, 285
 JTextField 268, 285
 JToolBar 268

JTree 268, 288, 298
 JWindow 260

K

KeyListener 265
 keySet() 187
 Klassen 82
 – abstrakt 83, 110
 – generisch 113
 – innere 95
 – ist-Beziehung 9
 Klassenhierarchie
 – Eingabe 211
 – Ereignisse 264
 Kommentare 12
 Konstruktor 85, 87, 89, 102
 – private 139
 Konvertierung 90
 Kopie
 – flache Kopie 58
 – tiefe Kopie 58

L

Label 268
 Lambda-Ausdruck 127
 last() 183
 lastIndexOf() 177
 lastKey() 189
 Layout 270
 LayoutManager 270
 length 55
 length() 153, 155
 limit() 195
 lines() 193
 LinkedList 172, 173
 List 172, 268
 List<E> 177
 ListIterator 178
 ListIterator<E> 177, 179
 LocalDate 157, 158
 LocalDateTime 157, 159
 localhost 333
 LocalTime 157, 159
 log() 157
 Logging 167
 logische Ausdrücke 70
 lokale Variable 17
 long 14
 Long 96
 LongStream 193
 lookingAt() 165

M

main 5
 map() 195
 Map 172
 Map.Entry<K,V> 187
 Map<K,V> 184, 185
 matches() 165
 Math.E 157
 Math.PI 157
 Matrix 66
 Mausbewegungen bearbeiten 309
 Mausclick bearbeiten 307
 max() 157, 193, 195
 Mengen
 - Differenz 181
 - Durchschnitt 181
 - Test auf leere Menge 181
 - Vereinigung 181
 Mengendiagramm 81
 MenuContainer 286
 Metadaten 162, 368
 Methode 43
 - Referenz 126
 Methoden 85
 min() 157, 193
 minus() 158
 modale Dialoge 316
 Model 269, 289
 Model-View-Controller 269
 Monitor 241, 252
 Month 159
 MouseAdapter 266
 MouseListener 265
 MouseMotionListener 265
 mousePressed 309
 MVC 269

N

new 56
 newInstance() 132
 nextDouble() 20
 nextInt() 20
 nichtmodale Dialoge 316
 NORTHEAST 275
 NORTHWEST 275
 notify() 243
 notifyAll() 243
 now() 158
 null 56
 NullPointerException 56

O

Oberklasse 84, 98, 106
 Object 98, 152
 ObjectInputStream 212
 ObjectOutputStream 214
 Objektorientierung 77
 objektrelationale Abbildung 364
 ODBC 351
 of(...) 196
 of() 158
 Optional 196
 Organisationsmodelle 234
 OutputStream 205, 206, 214
 OutputStreamWriter 206

P

package 140
 paint() 10
 Panel 268
 parallel() 193, 229
 parallele Server-Anwendung 335
 Parameter 44, 90
 - aktuell 44
 - formal 43, 44
 - Funktionen als 130
 - Variable Anzahl von Argumenten 97
 parse() 158
 Path 208
 - createDirectory 209
 - deleteIfExists 209
 - walkFileTree 209
 PATH 383
 Pattern 164
 Period 159
 Pipeline-Modell 234
 plus() 158
 plusDays(...) 159
 Polymorphie 81, 100, 104
 P-Operation 249
 Postcondition 145
 pow() 157
 Precondition 145
 Predicate 125, 196
 PreparedStatement 356, 357
 Primärschlüssel 354
 printf() 214, 216
 PrintStream 206, 214
 PrintWriter 206, 216, 219
 private 87, 142
 process(...) 244
 properties-Datei 362

protected 104, 142
 Prozess 232
 public 142
 put() 186
 putAll() 187

Q

quadratische Gleichung 28
 Quadratwurzel 36

R

Radiobutton 268
 RAM 13
 random() 157, 257
 range 193
 read() 207, 211
 readAllLines 209
 readLine() 207
 Reader 206
 ready() 207
 receive() 337
 reduce(...) 194, 196
 ReentrantReadWriteLock 249
 Referenzsemantik 91
 Reflection 132
 Reflexion 160
 – der Datenbank 350
 Registratur 346
 Reguläre Ausdrücke 164
 Rekursion 48
 RELATIVE 274
 Remote 346
 remove() 177, 180, 186
 repaint() 310
 replace() 153
 Reservierte Namen 24
 ResourceBundle 362
 ResultSet 353, 359, 367
 ResultSetMetaData 361, 368
 return 44
 reverse() 189
 rint() 157
 rmiregistry 345
 rollback() 356
 round() 157
 run() 231
 Runnable 230, 232
 RuntimeException 109

S

SAX 378, 379
 Scanner 20
 schemagen 376
 Schnittstellen 116
 – default-Methode 121
 – funktional 125
 – statische Methode 121
 Scrollbar 268
 Semaphoren 249
 send() 337
 Serialisierung 142
 Serializable 142, 160
 set() 177
 Set 171, 172
 setCharAt() 155
 setColor() 35
 setJMenuBar 286
 setLayout() 270
 setProperty() 156
 setRolloverIcon() 279
 setSavepoint() 356
 setToolTipText() 279
 short 14
 Short 96
 shuffle() 189
 SimpleFileVisitor 222
 sin() 156
 size() 180, 186
 sleep() 239
 Sockets 332
 sorted() 196, 198
 SortedMap 172, 188
 SortedSet 171, 182
 Sortieren 63
 – Bubble-Sort 63
 – Klasse java.util.Arrays 66
 – Quick-Sort 64
 SOUTHEAST 275
 SOUTHWEST 275
 split() 153
 sqrt() 157
 start() 240
 startsWith() 153
 static 22, 43, 88, 92
 – Initialisierung 93
 StAX 374
 stream() 193
 Stream 194
 String 16, 152
 StringBuffer 155
 StringBuilder 152, 154

Struktogramme 25
 sublist 178
 subMap() 188
 subSequence() 155
 subSet() 183
 substring() 153, 155
 Suche
 - binäre Suche 63
 - Ersetzen 166
 - Klasse java.util.Arrays 66
 sum() 193
 summingInt(...) 196
 Supplier 125, 196
 Swing 259
 SwingWorker 244
 switch 29
 synchronized 191, 241
 synchronizedList 173, 191
 synchronizedMap 191
 synchronizedSet 191
 synchronizedSortedMap 191
 synchronizedSortedSet 191
 Syntaxdiagramme 41
 System 152, 156

T

TableModel 294
 tailMap() 189
 tailSet() 183
 tan() 157
 TCP/IP 332, 333
 Team-Modell 234
 TemporalAdjusters 157, 159
 TextArea 268
 TextComponent 268
 Textdatei 220
 TextField 268
 TextListener 265
 this 85, 95
 Thread 152, 232
 - bereit 239
 - Event 262
 - fertig 239
 - Klasse Thread 230
 - laufend 239
 - vorhanden 239
 - wartend 239
 throw 39
 to() 158
 toCharArray() 153
 toDegrees() 157
 toList() 196

toLowerCase() 153
 toRadians() 157
 toString() 155
 toUpperCase() 153
 transient 142
 TreeMap 172, 185
 TreeSelectionListener 303
 TreeSet 172, 180
 trim() 153
 try 39
 - mit Ressourcen 109
 Türme von Hanoi 50, 321
 Turtle-Grafik 311
 Type-Cast 22
 Typumwandlungen 22

U

Überladen 47
 Überschreiben 99
 Übersetzer 2
 UDP 332
 UML 79
 UnaryOperator 125, 196
 Unboxing 96
 Unterbrechung 242
 Unterklasse 81, 98
 URI 379
 URL 332, 379
 URN 379

V

valueOf() 153
 values() 187
 Variable 14
 - Gültigkeitsbereich 23
 Vector 180
 Verbraucher 243
 Vererbung 81, 84
 - Design für 109
 - Diamant 122
 Vergleich 69
 Verteiler-Arbeiter-Modell 234
 VERTICAL 275
 Verzeichnis 222
 View 269, 289
 virtuelle Methoden-Tabelle 104
 Void 152
 V-Operation 249
 Vorbedingung 145
 Vorrangstufen 68

W

wait() 242
weightx 275
weighty 275
Wertzuweisung 17
- Objekte 90
while 32, 36
Window 260
WindowListener 265
with() 158
write() 215
Writer 206

X

xjc 376
x-Koordinate 10
XML 373
XSD 375

Y

yield() 240
y-Koordinate 10

Z

ZIP-Archiv 223
ZipFile 223
Zusicherung 144
Zustände von Threads 239