



Elisabeth
Jung

Java 9

Das Übungsbuch

Über 200 Aufgaben mit vollständigen Lösungen



Hinweis des Verlages zum Urheberrecht und Digitalen Rechtemanagement (DRM)

Der Verlag räumt Ihnen mit dem Kauf des ebooks das Recht ein, die Inhalte im Rahmen des geltenden Urheberrechts zu nutzen. Dieses Werk, einschließlich aller seiner Teile, ist urheberrechtlich geschützt. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtsgesetzes ist ohne Zustimmung des Verlages unzulässig und strafbar. Dies gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen und Einspeicherung und Verarbeitung in elektronischen Systemen.

Der Verlag schützt seine ebooks vor Missbrauch des Urheberrechts durch ein digitales Rechtemanagement. Bei Kauf im Webshop des Verlages werden die ebooks mit einem nicht sichtbaren digitalen Wasserzeichen individuell pro Nutzer signiert.

Bei Kauf in anderen ebook-Webshops erfolgt die Signatur durch die Shopbetreiber. Angaben zu diesem DRM finden Sie auf den Seiten der jeweiligen Anbieter.

Elisabeth Jung

Java 9 Das Übungsbuch

Über 200 Aufgaben mit
vollständigen Lösungen



mitp

Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

ISBN 978-3-95845-648-8

I. Auflage 2017

www.mitp.de

E-Mail: mitp-verlag@sigloch.de

Telefon: +49 7953 / 7189 - 079

Telefax: +49 7953 / 7189 - 082

© 2017 mitp Verlags GmbH & Co. KG, Frechen

Dieses Werk, einschließlich aller seiner Teile, ist urheberrechtlich geschützt. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtsgesetzes ist ohne Zustimmung des Verlages unzulässig und strafbar. Dies gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen und die Einspeicherung und Verarbeitung in elektronischen Systemen.

Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften.

Lektorat: Sabine Schulz

Sprachkorrektorat: Petra Heubach-Erdmann

Fachkorrektorat: Uwe Rozanski

Coverbild: © Mak @ fotolia.com

Satz: III-satz, Husby, www.drei-satz.de

Inhaltsverzeichnis

	Einleitung	25
I	Klassendefinition und Objektinstanziierung	33
1.1	Klassen und Objekte	33
	☆ Aufgabe 1.1: Definition einer Klasse	36
	☆ Aufgabe 1.2: Objekt (Instanz) einer Klasse erzeugen ...	37
1.2	Die Member einer Klasse: Felder und Methoden	37
	☆ Aufgabe 1.3: Zugriff auf Felder	38
	☆☆ Aufgabe 1.4: Aufruf von Methoden	38
1.3	Das Überladen von Methoden	39
	☆ Aufgabe 1.5: Eine Methode überladen.	39
1.4	Die Datenkapselung, ein Prinzip der objektorientierten Programmierung	40
	☆ Aufgabe 1.6: Zugriffsmethoden	40
1.5	Das »aktuelle Objekt« und die »this-Referenz«	40
	☆☆ Aufgabe 1.7: Konstruktordefinitionen	41
1.6	Die Wert- und Referenzübergabe in Methodenaufrufen	41
	☆☆☆ Aufgabe 1.8: Wertübergabe in Methoden (»call by value«).....	42
1.7	Globale und lokale Referenzen.	42
	☆☆ Aufgabe 1.9: Der Umgang mit Referenzen	42
	☆☆☆ Aufgabe 1.10: Wiederholungsaufgabe	43
1.8	Selbstreferenzierende Klassen und Felder (»self-referential classes and fields«)	44
	☆☆☆ Aufgabe 1.11: Der Einsatz von selbstreferenzierenden Feldern.....	44
1.9	Java-Pakete	45
	☆ Aufgabe 1.12: Die package-Anweisung	46
	☆ Aufgabe 1.13: Die import-Anweisung	47
1.10	Die Modifikatoren für Felder und Methoden in Zusammenhang mit der Definition von Paketen	47
	☆☆ Aufgabe 1.14: Pakete und die Sichtbarkeit von Members einer Klasse	47

1.11	Standard-Klassen von Java	48
	☆☆☆ Aufgabe 1.15: Aufruf von Methoden der Klasse Math	50
	☆☆ Aufgabe 1.16: Wiederholungsaufgabe	51
1.12	Die Wrapper-Klassen von Java und das Auto(un)boxing	52
	☆☆ Aufgabe 1.17: Die Felder und Methoden von Wrapper-Klassen	53
	☆ Aufgabe 1.18: Das Auto(un)boxing	53
1.13	Arrays (Reihungen) und die Klassen Array und Arrays	54
	☆☆ Aufgabe 1.19: Der Umgang mit Array-Objekten	55
1.14	Zeichenketten und die Klasse String	56
	☆ Aufgabe 1.20: Der Umgang mit String-Objekten	56
1.15	Mit der Version 5.0 eingeführte Spracherneuerungen für Arrays und Methoden	56
	☆ Aufgabe 1.21: Einfache und erweiterte for-Schleifen	57
	☆ Aufgabe 1.22: Methoden mit variablen Argumentenlisten	58
1.16	Das Initialisieren von Klassen- und Instanzfeldern	58
	☆ Aufgabe 1.23: Das Initialisieren von Instanzfeldern	59
	☆ Aufgabe 1.24: Das Initialisieren von Klassenfeldern	59
1.17	Private Konstruktoren	59
	☆ Aufgabe 1.25: Ein Objekt mit Hilfe eines privaten Konstruktoren erzeugen	60
	☆☆ Aufgabe 1.26: Mehrere konstante Werte (Objekte) mit Hilfe eines privaten Konstruktoren erzeugen	60
1.18	Lösungen	61
	Lösung 1.1	61
	Lösung 1.2	61
	Lösung 1.3	61
	Lösung 1.4	63
	Lösung 1.5	65
	Lösung 1.6	66
	Lösung 1.7	67
	Lösung 1.8	69
	Lösung 1.9	71
	Lösung 1.10	73
	Lösung 1.11	78
	Lösung 1.12	80
	Lösung 1.13	80
	Lösung 1.14	81

	Lösung I.15	82
	Lösung I.16	84
	Lösung I.17	87
	Lösung I.18	89
	Lösung I.19	92
	Lösung I.20	95
	Lösung I.21	97
	Lösung I.22	98
	Lösung I.23	100
	Lösung I.24	102
	Lösung I.25	103
	Lösung I.26	104
2	Abgeleitete Klassen und Vererbung	107
2.1	Abgeleitete Klassen	107
2.2	Die Konstruktoren von abgeleiteten Klassen	107
2.3	Abgeleitete Klassen und die Sichtbarkeit von Feldern und Methoden	107
	☆☆ Aufgabe 2.1: Test von Sichtbarkeits Ebenen	108
2.4	Das Verdecken von Klassenmethoden und das statische Binden von Methoden	109
	☆☆ Aufgabe 2.2: Der Aufruf von verdeckten Klassenmethoden	110
2.5	Das Überschreiben von Instanzmethoden und das dynamische Binden von Methoden	110
	☆ Aufgabe 2.3: Das dynamische Binden von Methoden ...	111
2.6	Vererbung und Komposition	111
	☆ Aufgabe 2.4: Die Komposition	112
	☆ Aufgabe 2.5: Die Vererbung	112
2.7	Kovariante Rückgabetypen in Methoden	113
	☆☆ Aufgabe 2.6: Die Benutzung von kovarianten Rückgabetypen	113
2.8	Verdeckte Felder	114
	☆☆☆ Aufgabe 2.7: Wiederholungsaufgabe	114
2.9	Vergrößernde und verkleinernde Konvertierung (»up- und down-casting«)	116
	☆☆☆ Aufgabe 2.8: Up- und Down-Casts	116
	☆☆ Aufgabe 2.9: Der Unterschied zwischen »ist-ein-« und »hat-ein- Beziehungen«	117

2.10	Der Polymorphismus, ein Prinzip der objektorientierten Programmierung	118
	☆☆☆ Aufgabe 2.10: Der »Subtyp-Polymorphismus« im Kontext einer Klassenhierarchie	119
2.11	Lösungen	120
	Lösung 2.1	120
	Lösung 2.2	123
	Lösung 2.3	125
	Lösung 2.4	127
	Lösung 2.5	129
	Lösung 2.6	130
	Lösung 2.7	133
	Lösung 2.8	137
	Lösung 2.9	141
	Lösung 2.10	144
3	Abstrakte Klassen und Interfaces	147
3.1	Abstrakte Klassen	147
3.2	Abstrakte Java-Standard-Klassen und eigene Definitionen von abstrakten Klassen	147
	☆☆ Aufgabe 3.1: Die abstrakte Klasse Number und ihre Unterklassen	147
	☆☆ Aufgabe 3.2: Definition einer eigenen abstrakten Klasse	148
3.3	Die Referenzen vom Typ einer abstrakten Klassen	149
	☆☆ Aufgabe 3.3: Der Subtyp-Polymorphismus für Methoden im Kontext einer Klassenhierarchie mit abstrakten Klassendefinitionen	149
	☆☆ Aufgabe 3.4: Der Subtyp-Polymorphismus für Felder im Kontext einer Klassenhierarchie mit abstrakten Klassendefinitionen	149
3.4	Interfaces (Schnittstellen)	150
	☆☆ Aufgabe 3.5: Die Definition eines Interface	150
3.5	Die Entscheidung zwischen abstrakten Klassen und Interfaces	151
	☆☆☆ Aufgabe 3.6: Paralleler Einsatz von Interfaces und abstrakten Klassen	152
3.6	Oberinterfaces	153
	☆☆ Aufgabe 3.7: Das Ableiten von Interfaces	153
3.7	Implementieren von mehreren Interfaces für eine Klasse	154
	☆☆ Aufgabe 3.8: Wiederholungsaufgabe	154

3.8	Die Vererbung an Beispielen von Java-Standard-Klassen und Standard-Interfaces	155
3.9	Das Klonen von Objekten	156
	☆ Aufgabe 3.9: Das Klonen von Instanzen der eigenen Klasse	156
	☆ Aufgabe 3.10: Das Klonen von Instanzen anderer Klassen	157
	☆☆ Aufgabe 3.11: Das Klonen von Arrays	157
	☆ Aufgabe 3.12: Das Überschreiben der clone()-Methode in Java 5.0	157
3.10	Die Gleichheit von Objekten	158
	☆ Aufgabe 3.13: Die Gleichheit von geklonten Objekten	158
3.11	Das oberflächliche und das tiefe Klonen (»shallow und deep cloning«).	159
	☆ Aufgabe 3.14: Das Klonen und der Copy-Konstruktor	159
	☆☆☆ Aufgabe 3.15: Tiefes Klonen am Beispiel von Array-Objekten	160
	☆☆☆ Aufgabe 3.16: Oberflächliches und tiefes Klonen für Referenztypen	160
3.12	Der Garbage Collector und das Beseitigen von Objekten	161
	☆ Aufgabe 3.17: Das Zerstören von Instanzen	161
3.13	Lösungen	162
	Lösung 3.1	162
	Lösung 3.2	163
	Lösung 3.3	164
	Lösung 3.4	165
	Lösung 3.5	167
	Lösung 3.6	169
	Lösung 3.7	172
	Lösung 3.8	173
	Lösung 3.9	177
	Lösung 3.10	178
	Lösung 3.11	178
	Lösung 3.12	179
	Lösung 3.13	180
	Lösung 3.14	181
	Lösung 3.15	183
	Lösung 3.16	185
	Lösung 3.17	187

4	Einführung in die graphische Programmierung	189
4.1	Das AWT (Abstract Windowing Toolkit) und Swing	189
4.2	Fenster unter graphischen Oberflächen	193
4.3	Die Klassen Graphics und Graphics2D	193
4.4	Methoden zum Zeichnen	194
	☆ Aufgabe 4.1: Eine einfache AWT-Komponente vom Typ Frame	197
	☆ Aufgabe 4.2: Eine Einfache Swing-Komponente vom Typ JFrame	197
	☆ Aufgabe 4.3: Ein JWindow-Fenster	198
	☆ Aufgabe 4.4: Ein JDialog-Fenster	198
	☆☆ Aufgabe 4.5: Das Neuzeichnen einer Swing-Komponente ohne Benutzung des Clip-Rectangles.	198
	☆☆ Aufgabe 4.6: Das Neuzeichnen einer Swing-Komponente mit Benutzung des Clip-Rectangles	199
	☆☆ Aufgabe 4.7: Das Parametrisieren der paint()-Methode.	199
	☆ Aufgabe 4.8: Die Instanzen der Klasse Color	200
4.5	Die Transparenz-Eigenschaft und der Hintergrund von Komponenten	200
4.6	Layout-Manager	201
	☆ Aufgabe 4.9: Die vordefinierten Layout-Manager für Standard-Klassen	203
	☆☆☆ Aufgabe 4.10: Die Layout-Manager von AWT-Komponenten	203
	☆☆☆ Aufgabe 4.11: Die Layout-Manager von Swing-Komponenten	204
	☆☆ Aufgabe 4.12: Das BorderLayout	204
	☆☆☆ Aufgabe 4.13: Das GridBagLayout	205
	☆☆ Aufgabe 4.14: Das null-Layout	206
4.7	Das Überlappen von Komponenten	206
	☆ Aufgabe 4.15: Das Verhalten von AWT-LW- und -HW-Komponenten	207
	☆☆ Aufgabe 4.16: AWT-Container mit LW- und HW-Kindkomponenten	207
	☆☆☆ Aufgabe 4.17: Z-Order-Index für AWT-LW-Komponenten.	208
	☆ Aufgabe 4.18: Das Verhalten von Swing-Komponenten	208
	☆☆☆ Aufgabe 4.19: Swing-Container und der Z-Order-Index für Swing-Komponenten	209

4.8	Das System als Auslöser für Zeichenoperationen	209
	☆☆ Aufgabe 4.20: Resizing von AWT-Komponenten	210
	☆☆ Aufgabe 4.21: Resizing von Swing-Komponenten	210
4.9	Eventbehandlung	211
4.10	Events auf niedriger Ebene	212
	☆ Aufgabe 4.22: Das Interface WindowListener und die Klasse WindowEvent	212
	☆ Aufgabe 4.23: Die Klasse WindowAdapter	212
4.11	Events auf höherer Ebene	213
	☆ Aufgabe 4.24: Das Interface ActionListener und die Klasse(ActionEvent)	213
	☆ Aufgabe 4.25: Das Interface KeyListener und die Klasse KeyEvent	213
4.12	Das Delegationsmodell in der Eventbehandlung	214
	☆☆ Aufgabe 4.26: Die Ereignisbehandlung in einer separaten Klasse definieren	214
4.13	Lösungen	215
	Lösung 4.1	215
	Lösung 4.2	216
	Lösung 4.3	217
	Lösung 4.4	218
	Lösung 4.5	219
	Lösung 4.6	220
	Lösung 4.7	222
	Lösung 4.8	223
	Lösung 4.9	225
	Lösung 4.10	226
	Lösung 4.11	229
	Lösung 4.12	232
	Lösung 4.13	233
	Lösung 4.14	235
	Lösung 4.15	236
	Lösung 4.16	237
	Lösung 4.17	239
	Lösung 4.18	240
	Lösung 4.19	242
	Lösung 4.20	245
	Lösung 4.21	247
	Lösung 4.22	249

	Lösung 4.23	250
	Lösung 4.24	251
	Lösung 4.25	252
	Lösung 4.26	253
5	Erweiterte graphische Programmierung	255
5.1	Der RootPane-Container	255
	☆☆ Aufgabe 5.1: Die LayeredPane einer Fensterkomponente	257
	☆☆ Aufgabe 5.2: Eine beliebige Instanz der Klasse JLayeredPane	258
	☆☆☆ Aufgabe 5.3: Das Positionieren von Komponenten in und innerhalb von Ebenen	259
	☆☆☆ Aufgabe 5.4: Eine GlassPane, die Ereignisse empfängt	259
	☆☆☆ Aufgabe 5.5: Eine benutzerdefinierte GlassPane-Komponente	260
5.2	Interne Fenster	260
	☆ Aufgabe 5.6: Die Instanzen der Klasse JInternalFrame	261
	☆☆ Aufgabe 5.7: Die Verschachtelung von internen Fenstern	261
5.3	Die Applikation als Auslöser für Zeichenoperationen	261
	☆☆ Aufgabe 5.8: Aufruf der repaint()-Methode für AWT-Komponenten	262
	☆☆ Aufgabe 5.9: Aufruf der repaint()-Methode für Swing-Komponenten	262
	☆☆ Aufgabe 5.10: Die getGraphics()-Methode	263
	☆☆ Aufgabe 5.11: Die Methoden paint(), getGraphics() und repaint() gleichzeitig nutzen	263
	☆☆☆ Aufgabe 5.12: Weiterführendes Zeichnen (»incremental painting«)	263
5.4	Das Interface Shape	265
	☆ Aufgabe 5.13: Die Methode draw() der Klasse Graphics2D	265
	☆ Aufgabe 5.14: Wiederholungsaufgabe	266
	☆ Aufgabe 5.15: Wiederholungsaufgabe	266
5.5	Praxisnahe Zeichenvorgänge und Eventbehandlungen	266
	☆☆☆ Aufgabe 5.16: Wiederholungsaufgabe	266

5.6	Benutzerdefinierte Event-Objekte und Event-Listener	267
	☆☆☆ Aufgabe 5.17: Das Erweitern der Klasse EventObject und des Interface EventListener	268
5.7	Lösungen	269
	Lösung 5.1	269
	Lösung 5.2	270
	Lösung 5.3	272
	Lösung 5.4	274
	Lösung 5.5	276
	Lösung 5.6	278
	Lösung 5.7	279
	Lösung 5.8	281
	Lösung 5.9	282
	Lösung 5.10	283
	Lösung 5.11	284
	Lösung 5.12	286
	Lösung 5.13	288
	Lösung 5.14	290
	Lösung 5.15	291
	Lösung 5.16	293
	Lösung 5.17	297
6	Das Erscheinungsbild einer Anwendung mit graphischer Oberfläche	303
6.1	Die Architektur Model View Controller (MVC) von Swing-Komponenten.	303
6.2	Benutzerdefinierte Modelle, die Standard-Model-Interfaces implementieren.	303
	☆☆ DefaultComboBox-Modell	304
	☆☆☆ Aufgabe 6.2: Benutzerdefiniertes ComboBox- Modell ohne Eventbehandlung	305
	☆☆☆ Aufgabe 6.3: Benutzerdefiniertes ComboBox- Modell mit Eventbehandlung	307
	☆☆☆ Aufgabe 6.4: Benutzung des ComboBox-Modells für eine Viewer-Komponente vom Typ JComboBox	307
	☆☆☆ Aufgabe 6.5: Die Vorlage für ein benutzerdefiniertes Modell als Erweiterung der Klasse PlainDocument definieren	308
6.3	Standard-Modelle am Beispiel der Klasse JTree	309

	☆	Aufgabe 6.6: Die Klassen JTree und DefaultMutableTreeNode	309
	☆☆☆	Aufgabe 6.7: Die Klassen DefaultTreeModel und DefaultTreeSelectionModel	309
	☆☆☆	Aufgabe 6.8: Benutzerdefiniertes Tree-Modell mit Eventbehandlung	310
6.4		Die UI-Delegationsklassen und ihre Instanzen, der UI-Delegate	311
6.5		Java-Standard-User-Interface-Delegationsklassen	312
	☆	Aufgabe 6.9: Der Standard-UI-Delegate	312
	☆	Aufgabe 6.10: Das Setzen der bevorzugten Größe für Komponenten	312
	☆☆	Aufgabe 6.11: Das Setzen einer minimalen Größe für Komponenten in Abhängigkeit von der Größe eines zu zeichnenden Bildes	313
	☆☆☆	Aufgabe 6.12: Das Setzen einer minimalen Größe für Komponenten in Abhängigkeit von der Größe der beim Zeichnen verwendeten Schrift	313
6.6		Benutzerdefinierte User-Interface-Delegaten	314
	☆☆☆	Aufgabe 6.13: Ein benutzerdefinierter UI-Delegate und das Setzen der bevorzugten Größe von Komponenten	314
	☆☆☆	Aufgabe 6.14: Ein benutzerdefinierter UI-Delegate für eine benutzerdefinierte Komponente	315
	☆☆☆	Aufgabe 6.15: Wiederholungsaufgabe	316
6.7		Die Klassen LookAndFeel, UIDefaults, UIManager und das Interface UIResource	317
	☆☆	Aufgabe 6.16: Lesen von Werten der UIDefaults-Tabelle	318
	☆	Aufgabe 6.17: Setzen von Werten der UIDefaults-Tabelle	319
	☆☆☆	Aufgabe 6.18: Setzen des UI-Delegationsobjektes mit der Methode put() der UIManager-Klasse	319
6.8		Standard-LookAndFeel-Komponenten	320
	☆☆☆	Aufgabe 6.19: LookAndFeel-spezifische Wiedergabe von Komponenten	321
	☆☆☆	Aufgabe 6.20: Wechseln zwischen allen installierten LookAndFeel-Komponenten	322
6.9		Das Erweitern der LookAndFeel-Klasse	322
	☆☆☆	Aufgabe 6.21: Benutzerdefinierte LookAndFeel-Komponenten	322

	☆☆☆ Aufgabe 6.22: Das Zusammenspiel zwischen den drei MVC-Komponenten Model, View und Controller	323
6.10	Lösungen	324
	Lösung 6.1	324
	Lösung 6.2	326
	Lösung 6.3	328
	Lösung 6.4	331
	Lösung 6.5	333
	Lösung 6.6	336
	Lösung 6.7	337
	Lösung 6.8	340
	Lösung 6.9	345
	Lösung 6.10	346
	Lösung 6.11	347
	Lösung 6.12	349
	Lösung 6.13	351
	Lösung 6.14	354
	Lösung 6.15	357
	Lösung 6.16	360
	Lösung 6.17	362
	Lösung 6.18	363
	Lösung 6.19	364
	Lösung 6.20	369
	Lösung 6.21	373
	Lösung 6.22	377
7	Innere Klassen	383
7.1	Die Definition von inneren Klassen und deren Instanzen	383
	☆☆ Aufgabe 7.1: Instanzieren von Member-Klassen innerhalb der umgebenden Klasse	384
	☆☆ Aufgabe 7.2: Instanzieren von Member-Klassen außerhalb der umgebenden Klasse	385
	☆☆ Aufgabe 7.3: Instanzieren von Static-Member-Klassen innerhalb der umgebenden Klasse	385
	☆☆ Aufgabe 7.4: Instanzieren von Static-Member-Klassen außerhalb der umgebenden Klasse	386
	☆☆ Aufgabe 7.5: Lokale Klassen	386
	☆☆ Aufgabe 7.6: Anonyme Klassen	387

	☆	Aufgabe 7.7: Member-Interfaces	387
	☆☆	Aufgabe 7.8: Member-Interface mittels einer anonymen Klasse implementieren	388
	☆☆	Aufgabe 7.9: Member-Interface von einer anderen Klasse implementieren.	388
	☆☆	Aufgabe 7.10: Member-Interfaces und Member-Klassen.	389
7.2		Innere Klassen am Beispiel von Event-Listener und Event-Adapter ...	389
	☆	Aufgabe 7.11: Den WindowAdapter als Member-Klasse definieren.	389
	☆	Aufgabe 7.12: Den WindowAdapter mittels einer anonymen Klasse implementieren	390
	☆	Aufgabe 7.13: Den ActionListener mittels einer anonymen Klasse implementieren	390
	☆☆☆	Aufgabe 7.14: Benutzdefinierte Event-Objekte und Event-Listener für JButton- und JTextField-Komponenten	390
	☆☆☆	Aufgabe 7.15: Benutzdefinierte Event-Objekte und Event-Listener für JRadioButton-Komponenten	391
	☆☆☆	Aufgabe 7.16: Benutzdefinierte Event-Objekte und Event-Listener für JLabel-Komponenten	391
7.3		Weitere Beispiele mit inneren Klassendefinitionen	392
	☆☆	Aufgabe 7.17: Die Methoden der Klasse JOptionPane aus einer anonymen Klasse aufrufen	393
	☆☆☆	Aufgabe 7.18: Einen Farbauswahldialog und die main()-Methode innerhalb von inneren Klassen definieren	393
	☆☆☆	Aufgabe 7.19: Wiederholungsaufgabe	394
7.4		Lösungen:	395
		Lösung 7.1	395
		Lösung 7.2	397
		Lösung 7.3	400
		Lösung 7.4	401
		Lösung 7.5	404
		Lösung 7.6	406
		Lösung 7.7	407
		Lösung 7.8	408
		Lösung 7.9	409
		Lösung 7.10	411

	Lösung 7.11	413
	Lösung 7.12	414
	Lösung 7.13	414
	Lösung 7.14	415
	Lösung 7.15	420
	Lösung 7.16	423
	Lösung 7.17	427
	Lösung 7.18	429
	Lösung 7.19	431
8	Generics	435
8.1	Die Generizität	435
8.2	Generische Klassen und Interfaces	436
	☆ Aufgabe 8.1: Generischer Datentyp als Behälter für die Instanzen vom Typ des Klassenparameters	437
	☆ Aufgabe 8.2: Generischer Datentyp als »Über-Typ« für die Instanzen vom Typ des Klassenparameters	437
	☆☆☆ Aufgabe 8.3: Generischer Stack	438
8.3	Wildcardtypen	438
	☆☆ Aufgabe 8.4: Ungebundene Wildcardtypen	439
	☆☆ Aufgabe 8.5: Obere Schranke (»upper bound wildcard«) für Wildcardtypen	439
	☆☆ Aufgabe 8.6: Untere Schranke (»lower bound wildcard«) für Wildcardtypen	440
8.4	Legacy Code, Erasure und Raw-Typen	441
	☆☆ Aufgabe 8.7: Raw-Typen am Beispiel einer generischen Klasse mit zwei Typparametern	442
	☆ Aufgabe 8.8: Generische Interfaces	443
	☆☆ Aufgabe 8.9: Brückenmethoden (»bridge methods«)	443
8.5	Generische Arrays	444
	☆☆ Aufgabe 8.10: Erzeugen von generischen Arrays	444
8.6	Generische Methoden	445
	☆☆☆ Aufgabe 8.11: Generische Methodendefinitionen	445
8.7	for-each-Schleifen für Collections	446
	☆☆☆ Aufgabe 8.12: Generische Arrays in generischen Methodendefinitionen	446
8.8	Generische Standard-Klassen und -Interfaces	447
	☆ Aufgabe 8.13: Die Klasse <code>ArrayList<E></code> und die Schnittstelle <code>List<E></code>	448

	☆☆	Aufgabe 8.14: Die Klasse <code>Vector<E></code> und die Schnittstelle <code>Collection<E></code>	448
	☆☆	Aufgabe 8.15: Die Klasse <code>TreeMap<K,V></code>	449
	☆☆☆	Aufgabe 8.16: Wiederholungsaufgabe	450
8.9		Enumerationen und die generische Klasse <code>Enum<E></code> extends <code>Enum<E></code>	450
	☆☆	Aufgabe 8.17: Die Definition von Enumerationen	451
	☆☆☆	Aufgabe 8.18: Konstruktoren und Methoden von <code>enum</code> -Klassen	451
8.10		Die Interfaces <code>Enumeration<E></code> , <code>Iterable<T></code> und <code>Iterator<E></code> sowie <code>Map<K,V></code> und <code>Set<E></code>	452
	☆☆	Aufgabe 8.19: Weitere generische Schnittstellen	452
8.11		Die Einträge der <code>UIDefaults</code> -Tabelle als Instanz der Klasse <code>Hashtable<Object, Object></code>	453
	☆☆	Aufgabe 8.20: Das Ändern der <code>font</code> -Eigenschaft von <code>Swing</code> -Komponenten	453
8.12		Die generischen Klassen <code>Class<T></code> und <code>Constructor<T></code> und das »dynamische« Erzeugen von Objekten	454
8.13		Das <code>Reflection</code> -API	454
	☆☆	Aufgabe 8.21: Die Klasse <code>Class<T></code>	458
	☆☆☆	Aufgabe 8.22: Die Klasse <code>Constructor<T></code>	459
	☆☆	Aufgabe 8.23: Erzeugen von generischen Arrays mit Hilfe eines <code>Class</code> -Objekts	460
	☆☆	Aufgabe 8.24: Mit <code>Reflection</code> Informationen zu Klassen, Oberklassen und Interfaces holen	461
	☆☆☆	Aufgabe 8.25: Das Interface <code>GenericDeclaration</code> und die Unterinterfaces von <code>Type</code>	462
8.14		Lösungen	464
		Lösung 8.1	464
		Lösung 8.2	465
		Lösung 8.3	466
		Lösung 8.4	468
		Lösung 8.5	469
		Lösung 8.6	471
		Lösung 8.7	472
		Lösung 8.8	476
		Lösung 8.9	477
		Lösung 8.10	478

	Lösung 8.11	479
	Lösung 8.12	481
	Lösung 8.13	482
	Lösung 8.14	483
	Lösung 8.15	485
	Lösung 8.16	486
	Lösung 8.17	488
	Lösung 8.18	489
	Lösung 8.19	492
	Lösung 8.20	495
	Lösung 8.21	496
	Lösung 8.22	498
	Lösung 8.23	501
	Lösung 8.24	502
	Lösung 8.25	507
9	Exceptions und Errors	515
9.1	Ausnahmen auslösen	515
9.2	Ausnahmen abfangen oder weitergeben	516
	☆ Aufgabe 9.1: Unbehandelte RuntimeExceptions	516
	☆ Aufgabe 9.2: Behandelte RuntimeExceptions	517
	☆☆ Aufgabe 9.3: Die Weitergabe von Ausnahmen	517
9.3	Das Verwenden von finally in der Ausnahmebehandlung	518
	☆ Aufgabe 9.4: Der finally-Block	518
	☆☆☆ Aufgabe 9.5: Geschachtelte try/catch-Blöcke	519
9.4	Ausnahmen manuell auslösen	520
	☆☆ Aufgabe 9.6: Standard-Ausnahmen manuell auslösen	520
9.5	Exception-Unterklassen erzeugen	521
	☆ Aufgabe 9.7: Benutzerdefinierte Ausnahmen manuell auslösen	521
	☆☆☆ Aufgabe 9.8: Wiederholungsaufgabe	521
9.6	Ketten von Ausnahmen	523
	☆☆☆ Aufgabe 9.9: Exception-Ketten	523
9.7	Die Ausnahmen bei einem Wechsel von LookAndFeel- Komponenten	524
	☆☆☆ Aufgabe 9.10: Die LookAndFeel-spezifischen Einträge der UIDefaults-Tabelle	524

9.8	Lösungen	525
	Lösung 9.1	525
	Lösung 9.2	526
	Lösung 9.3	527
	Lösung 9.4	529
	Lösung 9.5	530
	Lösung 9.6	534
	Lösung 9.7	536
	Lösung 9.8	538
	Lösung 9.9	542
	Lösung 9.10	545
10	Java-Typen	553
10.1	Typen in Java	553
10.2	Typprüfung und Typsicherheit mittels Generics	556
10.3	Subtyping für parametrisierte Typen	562
10.4	Die extends-Klausel	563
10.5	Typinferenz für Methoden	564
10.6	Typinferenz beim Erzeugen von Instanzen eines generischen Typs	565
10.7	Heap Pollution	567
10.8	Wildcard-Capture	569
	☆ Aufgabe 10.1: Typinferenz beim Instanzieren von generischen Klassen	570
	☆ Aufgabe 10.2: Der Diamond-Operator	571
	☆☆ Aufgabe 10.3: Schranken für Typvariablen und Typinferenz für Methoden	572
	☆☆☆ Aufgabe 10.4: Parametrisierte Typen und Wildcardtypen	575
	☆☆ Aufgabe 10.5: Generische Arraytypen	579
	☆☆ Aufgabe 10.6: Subtyping von Referenztypen	583
10.9	Multi-catch-Klausel und verbesserte Typprüfung beim Rethrowing von Exceptions	584
	☆ Aufgabe 10.7: Disjunction-Typ für Exceptions	585
	☆☆ Aufgabe 10.8: Typprüfung beim Rethrowing von Exceptions	587
10.10	Lösungen	589
	Lösung 10.1	589
	Lösung 10.2	593

	Lösung 10.3	598
	Lösung 10.4	604
	Lösung 10.5	621
	Lösung 10.6	627
	Lösung 10.7	634
	Lösung 10.8	636
II	Java 8 Lambdas und Streams	641
II.1	Mittels anonymer Klassen Code an Methoden übergeben.	641
II.2	Funktionale Interfaces	643
II.3	Syntax und Deklaration von Lambda-Ausdrücken	643
	☆ Aufgabe 11.1: Lambda-Ausdruck ohne Parameter versus anonymer Klasse	648
	☆ Aufgabe 11.2: Lambda-Ausdruck mit Parameter versus anonymer Klasse	651
	☆ Aufgabe 11.3: Weitere Beispiele mit anonymen Klassen und Lambda-Ausdrücken	651
II.4	Scoping und Variable Capture	652
	☆☆ Aufgabe 11.4: Die Umgebung von Lambda-Ausdrücken	653
	☆☆ Aufgabe 11.5: Die neuen funktionalen Interfaces Consumer<T> und Predicate<T> und die Übergabe von Lambda-Ausdrücken in Methoden.	654
	☆☆ Aufgabe 11.6: Wiederholungsaufgabe	656
II.5	Methoden- und Konstruktor-Referenzen.	659
	☆ Aufgabe 11.7: Methoden-Referenzen in Zuweisungen	661
	☆☆ Aufgabe 11.8: Methoden-Referenzen als Argumente in Methodenaufrufen übergeben	662
	☆☆ Aufgabe 11.9: Konstruktor-Referenzen und die neuen funktionalen Interfaces Supplier<T> und Function<T,R>	664
II.6	Default-Methoden und statische Methoden in Interfaces	666
II.7	Das neue Interface Stream	668
II.8	Die forEach-Methoden von Iterator, Iterable und Stream	672
II.9	Die Default-Methoden des Map-Interface	674
	☆ Aufgabe 11.10: Die forEach()-Methode von Iterable.	676
	☆ Aufgabe 11.11: Die forEach()-Methode des Iterator-Interface	677

	☆	Aufgabe 11.12: Die funktionalen Interfaces BiConsumer<T,U>, BiPredicate<T,U> und BiFunction<T,U,R>	678
	☆☆	Aufgabe 11.13: Die Methoden des Interface Stream und die Behandlung von Exceptions in Lambda-Ausdrücken.	680
	☆	Aufgabe 11.14: Die forEach- und replace-Methoden des Map-Interface	683
11.10		Das Interface Collector und die Klasse Collectors. Reduktion mittels Methoden von Streams und Kollektoren.	684
	☆☆	Aufgabe 11.15: Weitere Methoden des Interface Stream: limit(), count(), max(), min(), skip(), reduce() und collect()	689
	☆☆☆	Aufgabe 11.16: Das Interface Collector und die Klasse Collectors	696
11.11		Parallele Streams	698
11.12		Die Bulk-Operationen der Klasse ConcurrentHashMap	702
	☆☆☆	Aufgabe 11.17: Parallele Streams	705
	☆☆☆	Aufgabe 11.18 : Die neuen Methoden von ConcurrentHashMap	707
11.13		Lösungen	709
		Lösung 11.1.	709
		Lösung 11.2	716
		Lösung 11.3.	718
		Lösung 11.4	720
		Lösung 11.5.	725
		Lösung 11.6	729
		Lösung 11.7	738
		Lösung 11.8	739
		Lösung 11.9	749
		Lösung 11.10	754
		Lösung 11.11	755
		Lösung 11.12.	756
		Lösung 11.13.	760
		Lösung 11.14.	770
		Lösung 11.15.	774
		Lösung 11.16	795
		Lösung 11.17.	803
		Lösung 11.18.	818

12	Die neuen Features von Java 9	827
12.1	Einführung.	827
12.2	Factory-Methoden in Collections	827
	☆ Aufgabe 12.1: Factory-Methoden für List, Set und Map	828
12.3	Stream-Erweiterungen	829
	☆ Aufgabe 12.2: Die neuen Stream-Methoden.	830
12.4	Die map() und flatMap()-Methoden von Stream und Optional.	831
	☆☆ Aufgabe 12.3 : map() versus flatMap()	834
12.5	Optional-Erweiterungen	837
	☆☆ Aufgabe 12.4: Die neuen Methoden der Optional-Klasse	837
12.6	Kollektoren in Java 9	841
	☆☆ Aufgabe 12.5: Die filtering() und mapping()-Kollektoren	841
12.7	Process-API-Erweiterungen	843
	☆☆ Aufgabe 12.6: Die neuen Interfaces ProcessHandler und ProcessHandler.Info.	844
12.8	Reactive Programming	847
	☆☆ Aufgabe 12.7: Die Interfaces der Flow-API implementieren	849
	☆☆☆ Aufgabe 12.8: Ein Publisher, der zwei Subscriber beliefert	851
12.9	Das Java-Modulsystem	855
	☆ Aufgabe 12.9: Eine einfache Modul-Definition	863
	☆☆ Aufgabe 12.10: Eine Applikation mit mehreren Modulen	865
	☆☆ Aufgabe 12.11: Implizites Lesen von Modulen	867
	☆☆ Aufgabe 12.12: Eine modulbasierte Service-Implementierung	868
12.10	Platform-Logging	869
	☆☆ Aufgabe 12.13: Eine Default-Implementation von System.LoggerFinder	871
	☆☆ Aufgabe 12.14: Ein benutzerdefinierter System.Logger	874
12.11	Multi-Resolution Images	876
	☆☆ Aufgabe 12.15: Das Interface MultiResolutionImage und die Klasse BaseMultiResolutionImage	878
12.12	Die Java-Shell (JShell)	881
	☆ Aufgabe 12.16: Die JShell-API	886

12.13	Andere Änderungen aus dem JDK 9	887
	☆ Aufgabe 12.17: Der Diamond-Operator in Java 9	888
	☆ Aufgabe 12.18: Private Interface-Methoden	890
	☆ Aufgabe 12.19: Die try-with-resources-Anweisung.	894
12.14	Java 9 Internals	895
12.15	Lösungen	896
	Lösung 12.1	896
	Lösung 12.2	898
	Lösung 12.3	902
	Lösung 12.4	910
	Lösung 12.5	917
	Lösung 12.6	923
	Lösung 12.7	930
	Lösung 12.8	935
	Lösung 12.9	943
	Lösung 12.10	944
	Lösung 12.11	950
	Lösung 12.12	956
	Lösung 12.13	960
	Lösung 12.14	964
	Lösung 12.15	971
	Lösung 12.16	980
	Lösung 12.17	982
	Lösung 12.18	985
	Lösung 12.19	988
	Stichwortverzeichnis	991



Einleitung

Das Java 9 Übungsbuch ist aus der Erkenntnis entstanden, dass zu umfangreiche Beispiele mit komplizierten Algorithmen beim Lernen von Java am Anfang keine echte Hilfe bieten. Darum liegt der Schwerpunkt des Buches nicht auf der Umsetzung von komplizierten Vorgängen, sondern auf der Art und Weise, wie man die nicht immer für den Leser verständlich formulierten Erläuterungen zu Java-Klassen und -Interfaces aus der Dokumentation mit einfachen Beispielen erklären und gleichzeitig die zugrunde liegenden Konzepte damit erörtern kann.

Das Java 9 Übungsbuch wendet sich in erster Linie an Lehrer, Schüler und Studenten als Begleitliteratur zum Lernen der Programmiersprache Java, ist aber auch zum Selbststudium für alle Interessenten an dem Erlernen der Programmiersprache geeignet.

Durch die Einfachheit und Vollständigkeit der Aufgabenlösungen sowie die unterschiedlichen Lösungsmöglichkeiten erhält der Leser ein fundiertes Verständnis für die Aufgabenstellungen und deren Lösungen.

Durch das Lösen von Aufgaben soll der in Referenz- und Lehrbüchern von Java angebotene Stoff vertieft werden, und die dabei erzielten Ergebnisse können anhand der Lösungsvorschläge überprüft werden. Die Beispiele im Buch sind eher selten von zu komplexer Natur, sodass der eigentliche Zweck nicht in den Hintergrund tritt, und alle beschriebenen Themen können tiefgehend und präzise damit eingeübt werden.

Vorkenntnisse

Es ist Voraussetzung, dass der Leser zusätzlich mit einem Lehrbuch zu Java arbeitet bzw. bereits damit gearbeitet hat. Die grundlegenden Erläuterungen zu Java in diesem Buch können lediglich als Wiederholung des bereits vorhandenen Wissens dienen, reichen aber nicht aus, um die Sprache Java erst neu zu lernen.

Als weitere Voraussetzung gelten Grundlagen im Bereich der Programmierung und im Umgang mit dem Betriebssystem. Ein paralleler Zugriff auf die Java-Online-Dokumentation kann Hilfe zu den Java-Standard-Klassen bieten.

Aufbau des Buches

Jedes Kapitel beginnt mit einer kurzen und knappen Wiederholung des Stoffes, der in den Übungsaufgaben dieses Kapitels verwendet wird. Danach folgen alle Aufgabenstellungen der Übungen. Am Ende des Kapitels folgen gesammelt alle Lösungen der Übungsaufgaben mit Kommentaren, Erläuterungen und Hinweisen.

Alle Kapitel enthalten Wiederholungsaufgaben, in denen versucht wird, die bis zur angelangten Stelle definierten Begriffe zusammenzutragen.

Die Aufgaben haben unterschiedliche Schwierigkeitsgrade. Dieser wird im Aufgabenkopf durch 1 bis 3 Sternchen gekennzeichnet:



1 Sternchen für besonders einfache Aufgaben, die auch von Anfängern leicht bewältigt werden können;



2 Sternchen für etwas kompliziertere Aufgaben, die einen durchschnittlichen Aufwand benötigen;



3 Sternchen für Aufgaben, die sich an geübte Programmierer richten und einen wesentlich höheren Aufwand oder die Kenntnis von speziellen Details erfordern.

Die Programme aus früheren Übungen werden teilweise in späteren Übungen gebraucht und es wird auch immer wieder auf theoretische Zusammenhänge zurückgekommen oder hingewiesen.

Die Lösungsvorschläge verfügen über umfangreiche Kommentare, sodass ein Verständnis für die durchgeführte Aufgabe auch daraus abgeleitet werden kann und dies jede einzelne Aufgabe im Gesamtkontext unabhängig erscheinen lässt.

In den Kapiteln 1, 2 und 3 liegt das Hauptmerkmal auf den Eigenheiten der objektorientierten Programmierung mit Java. Durch eine Vielzahl von Beispielen wird gezeigt, was die Java-Standard-Klassen an Funktionalitäten bieten und wie diese sinnvoll in die Definition von eigenen Klassen eingebettet werden können.

Die Kapitel 4 bis 6 erproben umfangreiche Kenntnisse über die Java-GUI-Programmierung mit AWT und Swing.

Die darauf folgenden Spezialkapitel über Innere Klassen, Generics und Exceptions bringen parallel zu den Aufgaben, die einer umfangreichen Darstellung von neu

eingeführten Begriffen und Definitionen dienen, auch weiterführende Aufgabenbeispiele zu den bisherigen Kapiteln.

Weil der Schwerpunkt des Buches nicht auf der Umsetzung von aufwendigen Algorithmen liegen soll, werden einfache Beispiele mit Zahlen, Buchstaben, Wörtern, Büchern, Wochentagen, geometrischen Figuren etc. und teilweise auch mit ganz abstrakten Klassennamen wie `Klasser`, `Klassez`, `KlasseA`, `KlasseB` etc. verwendet. In den Beispielen zur graphischen Programmierung werden nicht alle Komponenten behandelt – dies machen ja vorrangig Referenzbücher – sondern die Beispiele dienen dazu, die Basiskonzepte von Swing-Architekturen in sich ergänzend zu beschreiben und können ganz einfach auch für andere Swing-Komponenten umgesetzt werden.

Das Kapitel 10 wurde um Erweiterungen zu den Themen Java-Typen, Generizität und Reflection ergänzt, um eine bessere Abgrenzung von Begriffen und Interpretationen, die sich im Laufe der Jahre etabliert haben, zu erreichen. Neue Aufgaben, die der Erklärung von Typinferenz, Subtyping von parametrisierten und Wildcardparametrisierten Typen, Heap-Pollution und Wildcard-Capture dienen sollen, sind hinzugekommen. Andere wiederum demonstrieren die Benutzung der Interfaces des Reflection-API in der Abfrage von generischen und nichtgenerischen Informationen zu Klassenstrukturen.

Mit der im März 2014 freigegebenen Java-Version 8.0 eröffnen sich ganz neue Betrachtungsweisen und Programmier Techniken in der Entwicklung von Applikationen mit Java, die in Kapitel 11 präsentiert werden.

Eine der wichtigsten Neuerungen in Java 8 sind neue Sprachmittel, die sogenannten Lambda-Ausdrücke, eine Art anonyme Methoden, die auf funktionalen Interfaces basieren. Diese besitzen jedoch eine viel kompaktere Syntax als Methoden. Das resultiert daraus, dass in ihrer Benutzung auf Namen, Modifikatoren, Rückgabotyp, `throws`-Klausel und in vielen Fällen auch auf Parameter verzichtet werden kann. Mit ihnen kann Funktionalität ausgeführt, gespeichert und übergeben werden, wie dies bisher nur von Instanzen in Java bekannt war.

Damit verbundene Themen wie die Gegenüberstellung zu anonymen Klassen, Syntax und Semantik, Behandlung von Exceptions, Scoping und Variable Capture, Methoden- und Konstruktor-Referenzen werden in den ersten Unterkapiteln des 11. Kapitels dieses Buches beschrieben und an Hand von vielen Beispielen erläutert.

Des Weiteren finden Sie hier die Beschreibung aller neuen funktionalen Interfaces und deren Methoden. Mit Java 8 wurden sogenannte Default-Methoden eingeführt. Diese werden in der Literatur auch als »virtual extension«- bzw. »defender«-Methoden bezeichnet und Schnittstellen, die über derartige Methoden verfügen, als erweiterte Schnittstellen. Damit können Interfaces zusätzlich zu abstrakten Methoden konkrete Methoden in Form von Standard-Implementierungen definieren und in Java wird die Mehrfachvererbung von Funktionalität ermöglicht. Neben Default-Methoden können Interfaces in Java 8 auch statische Methoden enthalten. Anders

als die statischen Methoden von Klassen werden diese jedoch nicht von abgeleiteten Typen geerbt.

Die nachfolgenden Unterkapitel beschäftigen sich im Detail mit der Definition und Nutzung von Streams. Ein Stream besteht aus einer Folge von Werten (in der Literatur wird auch von Sequenzen von Elementen gesprochen), die nur teilweise von mehreren in einer Pipeline dazwischenliegenden Operationen ausgewertet und durch eine abschließende Operation bereitgestellt werden. Diese Operationen werden in Java als Methodenaufrufe formuliert, die Funktionalität in Form von Lambdas und Methoden-Referenzen entgegennehmen können und diese auf alle Elemente der Folge anwenden.

Aufbauend auf Lambda-Ausdrücken ermöglichen diese eine ganz neue Arbeitsweise mit Arrays, Collections und Maps. Sie bringen vielversprechende Optimierungen mit sich, wie z.B. die Memoisation, eine Technik, um Programme zu beschleunigen, indem Rückgabewerte von Funktionen zwischengespeichert statt neu berechnet werden. Dazu gehören auch neue Parallelisierungstechniken, die auf den mit Java 7 eingeführten Fork/Join-Mechanismen basieren, interne Iterationen anstelle von den bisher bekannten externen benutzen und in Bibliotheken direkt integriert wurden. Als Beispiel im Umgang mit parallelen Streams wird im letzten Unterkapitel auf die Neuerungen aus der Klasse `ConcurrentHashMap` eingegangen.

Anstatt zu spezifizieren, »wie ein Ergebnis zu berechnen ist«, indem eine Datenstruktur von Anfang bis Ende durchlaufen wird und für jedes Element, das eventuell einer bestimmten Bedingung genügt, ein Wert berechnet und z.B. einer Summe hinzugefügt wird, braucht man mit Java 8 nur zu formulieren, »was zu erledigen ist«, wie z.B. das Berechnen der Summe oder eines maximalen/minimalen Werts von Elementen, die die erwünschte Bedingung erfüllen. Damit wird die Logik des Durchlaufens einer Datenstruktur von der Funktionalität, die auf jedem ihrer -Elemente durchgeführt wird, getrennt.

Mit einer Vielzahl von Aufgaben basierend auf Lambdas, Streams und Kollektoren (in denen Stream-Elemente angesammelt und reduziert werden können) werden die neuen Techniken aus Java 8 angewandt und alle neuen Begriffe erklärt.

In Kapitel 12 werden die Neuerungen zu Java 9 präsentiert. Aus der großen Anzahl von JEPs (JDK Enhancements Proposals), die in die neue Version eingeflossen sind, werden einige wichtige Bereiche, die die Themen aus diesem Buch ergänzen, und erste Schritte in der Modularisierung näher erläutert und wie immer anhand von Aufgaben eingeübt.

Die ersten Unterkapitel beschäftigen sich mit den neuen Factory-Methoden aus Collections, den Erweiterungen des Stream-Interface und der Optional-Klasse sowie mit den neuen Kollektoren von Java 9.

Java 9 bringt mehrere Updates in der Process-API mit sich, die dazu dienen sollen, auf einfachere Art und Weise als bisher Informationen über laufende Prozesse zu ermitteln, vor allem für den JVM-Prozess und selbst gestartete Prozesse. Auf diese

wird im Unterkapitel 12.7 Bezug genommen.

Das Unterkapitel 12.8 widmet sich der Unterstützung von Reactive Streams. Im Reactive-Programming-Model existieren ein Publisher und ein Subscriber. Der Publisher veröffentlicht einen Stream von Daten, an den der Subscriber asynchron angeschlossen ist. Die Verbindung, die zwischen einem Publisher und einem Subscriber aufgebaut wird, wird mit einer Subscription beschrieben. Wie die Beispiele aus diesem Unterkapitel zeigen, kann über den dazwischen geschalteten Vermittler gesteuert werden, dass die Subscriber Daten anfordern und die Publisher möglichst so viele Daten senden, wie von den Subscribern nicht-blockierend verarbeitet werden können.

Die mit Sicherheit wichtigste und umfangreichste Änderung, die Java 9 mit sich bringt, ist das neue Java-Modulsystem. Mit dem neuen Modulsystem wurde Java selbst modular gemacht und es können eigene Applikationen und Bibliotheken modularisiert werden.

Im Unterkapitel 12.9 werden im Detail beschrieben:

- Das Erzeugen von Modulen und deren Abhängigkeiten
- Das Packaging von Modul-Code
- Die Implementierung von Services
- Die Modularisierung der Java-Plattform

Mit einer Vielzahl von Aufgaben können die neuen Strukturen eingeübt werden.

In Ergänzung dazu werden in den Unterkapiteln 12.10 und 12.11 das Platform-Logging und Multi-Resolution-Images mittels modulbasierter Applikationen präsentiert.

Im JDK 9 ist mit dieser Version auch ein neuer Kommandointerpreter, die Java-Shell (JShell) zu finden. Sie ist ein einfaches Tool, mit dem kleine Java-Programme und einzelne Anweisungen, sogenannte Snippets, interaktiv getestet werden können.

Die JShell kann im Gegensatz zu JavaScript nicht als Skript-Sprache in Java integriert werden. Stattdessen verfügt diese über eine eigene API. Ein Beispiel dazu finden Sie ebenfalls in diesem Unterkapitel.

Weitere Änderungen aus dem JDK 9 wie:

- der Diamond-Operator für innere anonyme Klassen,
- private Methoden in Interfaces,
- Ergänzungen für `try-with-resources`-Blocks

sind Thema des Unterkapitels 12.13 und im letzten Unterkapitel 12.14 werden Java-Internals wie Compact-Strings und die neue Versionierung von Java angesprochen.

Alle in diesem Buch enthaltenen Beispiele von Java-Applikationen sind und bleiben auch ohne Modularisierung unter den nachfolgenden Java-Versionen ablaufbar und dienen in erster Linie dazu, die Java-Programmiersprache kennenzulernen. Sollen diese modularisiert werden, kann man sich nach den in der Datei `Java9Migration.pdf` präsentierten Empfehlungen richten, die zusätzlich zu dem gesamten Source-Code aus diesem Buch auf der Webseite des Verlags zum Download angeboten wird.

An dieser Stelle möchte ich auf das dem Buch zugrunde liegende Konzept hinweisen, dass parallel zu einfachen Aufgaben, die zu allen eingeführten Definitionen und Begriffen gebracht werden, auch Aufgaben von einem höheren Schwierigkeitsgrad präsentiert werden. Dabei werden anhand von inhaltlichen Zusammenhängen zwischen den Beispielen viele Basiskonzepte von Java erläutert.

Das Buch soll möglichst parallel zu einer Vielzahl von Java-Lehrbüchern eingesetzt werden können und einen Beitrag dazu leisten, die große Fülle von Informationen, die auf uns über die API-Dokumentation zukommt, besser einzuordnen und korrekt anwenden zu können.

Im Inhaltsverzeichnis werden alle Aufgaben mit einer Aufgabennummer und einer Kurzbeschreibung aufgelistet. Das Stichwortverzeichnis enthält alle wichtigen Begriffe sowie unter dem Oberbegriff »Klassen-« bzw. »Schnittstellendefinitionen« die Namen aller Beispiel-Klassen und -Interfaces.

Benötigte Software

Für die Durchführung der Übungen in den Kapiteln 1 bis 9 können Sie das Java Development Kit (JDK) der Java Standard Edition (SE) Version 5.x oder 6.x verwenden. Für die Übungen von Kapiteln 10 brauchen Sie die Version 7.0 oder höher, für die von Kapitel 11 die Version 8.0 oder höher und für die von Kapitel 12 die neue Version 9.

Das aktuelle Java Development Kit der Java Standard Edition können Sie sich kostenlos von der Java-Homepage von Oracle <http://www.oracle.com/technetwork/> herunterladen. Das JDK umfasst sowohl die Software zur Programmerstellung als auch das JRE (Java Runtime Environment) für die Programmausführung.

Graphische Entwicklungsoberflächen (wie z.B. Eclipse) sind keine Voraussetzung und auch nicht Bestandteil dieses Buches. Die Programme lassen sich grundsätzlich mit einem Texteditor wie z.B. Notepad++ oder auch Wordpad eingeben und über die Kommandozeile durch den Aufruf der Programme `javac`, `jar` und `java` übersetzen, paketieren und starten. Die vollständigen Programmaufrufe sind bei jeder Aufgabe angegeben.

Sollte Ihnen beim Erlernen der Programmiersprache selbst eine Entwicklungsumgebung nicht zu aufwändig oder unübersichtlich erscheinen, steht Ihnen nichts im Wege, die zu den Aufgaben zugehörigen Programm- und Klassendateien zum Testen oder auch Ergänzen in eine solche einzubetten.