

Inhaltsverzeichnis

	Einleitung	27
I	Klassendefinition und Objektinstanziierung	33
I.1	Klassen und Objekte	33
	☆ Aufgabe I.1: Definition einer Klasse	36
	☆ Aufgabe I.2: Objekt (Instanz) einer Klasse erzeugen ..	37
I.2	Die Member einer Klasse: Felder und Methoden	37
	☆ Aufgabe I.3: Zugriff auf Felder	38
	☆☆ Aufgabe I.4: Aufruf von Methoden	38
I.3	Das Überladen von Methoden	39
	☆ Aufgabe I.5: Eine Methode überladen	39
I.4	Die Datenkapselung, ein Prinzip der objektorientierten Programmierung	40
	☆ Aufgabe I.6: Zugriffsmethoden	40
I.5	Das »aktuelle Objekt« und die »this-Referenz«	40
	☆☆ Aufgabe I.7: Konstruktordefinitionen	41
I.6	Die Wert- und Referenzübergabe in Methodenaufrufen	41
	☆☆☆ Aufgabe I.8: Wertübergabe in Methoden (»call by value«)	42
I.7	Globale und lokale Referenzen	42
	☆☆ Aufgabe I.9: Der Umgang mit Referenzen	42
	☆☆☆ Aufgabe I.10: Wiederholungsaufgabe	43
I.8	Selbstreferenzierende Klassen und Felder (»self-referential classes and fields«)	44
	☆☆☆ Aufgabe I.11: Der Einsatz von selbstreferenzierenden Feldern	44
I.9	Java-Pakete	45
	☆ Aufgabe I.12: Die package-Anweisung	46
	☆ Aufgabe I.13: Die import-Anweisung	47
I.10	Die Modifikatoren für Felder und Methoden in Zusammenhang mit der Definition von Paketen	47
	☆☆ Aufgabe I.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 Sichtbarkeitssebenen	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
	☆☆ Aufgabe 6.1: Die AWT-Klasse List und ein 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, Viewer 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.II	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 Collectionen	446
	☆☆☆ Aufgabe 8.12: Generische Arrays in generischen Methodendefinitionen	446
8.8	Generische Standard-Klassen und -Interfaces	447
	☆ Aufgabe 8.13: Die Klasse ArrayList<E> und die Schnittstelle List<E>	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 extends 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 font-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		Definition von benutzerdefinierten Modellen, die generische Klassen und generische Interfaces benutzen	464
	☆☆☆	Aufgabe 8.26: Ein benutzerdefiniertes <code>UIDefaults</code> -Tree-Modell, das die Eigenschaften von <code>JButton</code> -, <code>JList</code> - und <code>JTree</code> -Komponenten speichert	464
	☆☆☆	Aufgabe 8.27: Die Syntax der <code>firexxx</code> -Methodensignaturen von <code>TreeModel</code> -Klassen	465
	☆☆☆	Aufgabe 8.28: Ein <code>UIDefaults</code> -Tree-Modell, das die <code>String</code> -Eigenschaften von <code>Swing</code> -Komponenten speichert	466
	☆☆☆	Aufgabe 8.29: Benutzerdefiniertes <code>List</code> -Modell ohne <code>Event</code> behandlung	467
	☆☆	Aufgabe 8.30: Benutzung des <code>List</code> -Modells für eine <code>Viewer</code> -Komponente vom Typ <code>JList</code>	468

	☆☆☆ Aufgabe 8.31: Benutzerdefiniertes List-Modell mit Eventbehandlung	468
	☆☆ Aufgabe 8.32: Benutzung des List-Modells für eine Viewer-Komponente vom Typ JList	469
	☆☆☆ Aufgabe 8.33: Vervollständigung der Lösung für ein benutzerdefiniertes List-Modell	469
8.15	Lösungen	470
	Lösung 8.1	470
	Lösung 8.2	471
	Lösung 8.3	472
	Lösung 8.4	474
	Lösung 8.5	475
	Lösung 8.6	477
	Lösung 8.7	478
	Lösung 8.8	482
	Lösung 8.9	483
	Lösung 8.10	484
	Lösung 8.11	485
	Lösung 8.12	487
	Lösung 8.13	488
	Lösung 8.14	489
	Lösung 8.15	491
	Lösung 8.16	492
	Lösung 8.17	494
	Lösung 8.18	495
	Lösung 8.19	498
	Lösung 8.20	501
	Lösung 8.21	502
	Lösung 8.22	504
	Lösung 8.23	507
	Lösung 8.24	508
	Lösung 8.25	513
	Lösung 8.26	520
	Lösung 8.27	524
	Lösung 8.28	528
	Lösung 8.29	532
	Lösung 8.30	534
	Lösung 8.31	536
	Lösung 8.32	539

	Lösung 8.33	541
9	Exceptions and Errors	547
9.1	Ausnahmen auslösen	547
9.2	Ausnahmen abfangen oder weitergeben	548
	☆ Aufgabe 9.1: Unbehandelte RuntimeExceptions	548
	☆ Aufgabe 9.2: Behandelte RuntimeExceptions	549
	☆☆ Aufgabe 9.3: Die Weitergabe von Ausnahmen	549
9.3	Das Verwenden von finally in der Ausnahmebehandlung	550
	☆ Aufgabe 9.4: Der finally-Block	550
	☆☆☆ Aufgabe 9.5: Geschachtelte try/catch-Blöcke	551
9.4	Ausnahmen manuell auslösen	552
	☆☆ Aufgabe 9.6: Standard-Ausnahmen manuell auslösen	552
9.5	Exception-Unterklassen erzeugen	553
	☆ Aufgabe 9.7: Benutzerdefinierte Ausnahmen manuell auslösen	553
	☆☆☆ Aufgabe 9.8: Wiederholungsaufgabe	553
9.6	Ketten von Ausnahmen	555
	☆☆☆ Aufgabe 9.9: Exception-Ketten	555
9.7	Die Ausnahmen bei einem Wechsel von LookAndFeel- Komponenten	556
	☆☆☆ Aufgabe 9.10: Die LookAndFeel-spezifischen Einträge der UIDefaults-Tabelle	556
	☆☆☆ Aufgabe 9.11: Wiederholungsaufgabe	557
9.8	Lösungen	558
	Lösung 9.1	558
	Lösung 9.2	559
	Lösung 9.3	560
	Lösung 9.4	561
	Lösung 9.5	563
	Lösung 9.6	567
	Lösung 9.7	568
	Lösung 9.8	570
	Lösung 9.9	574
	Lösung 9.10	577
	Lösung 9.11	585

10	Neue Features von Java 6.0	589
10.1	Ergänzungen im Sprachumfeld und in der Behandlung von Exceptions mit Java 6.0	589
	☆☆ Aufgabe 10.1: Der GLOBAL_LOGGER_NAME, Array-Copies, Empty-Strings und throw in einem catch-Block.	589
10.2	Die neuen displayName[s] Methoden der Klasse Calendar	590
	☆ Aufgabe 10.2: Die Klassen Calendar und Date.	591
10.3	Die Interfaces NavigableMaps und NavigableSets	592
	☆☆ Aufgabe 10.3: Die Methoden der Interfaces NavigableMap und NavigableSet.	592
10.4	Splash Screens	592
	☆ Aufgabe 10.4: Einen Splash Screen setzen.	593
	☆ Aufgabe 10.5: Auf einen Splash Screen zeichnen	593
10.5	Der Dialog-ModalityType	594
	☆☆☆ Aufgabe 10.6: Die Dialog.ModalityType-Option	595
10.6	Sortieren und Filtern von Tabelleneinträgen	596
	☆ Aufgabe 10.7: Das Sortieren von Tabelleneinträgen mit Hilfe der Klasse TableRowSorter<TableModel> ...	596
	☆☆ Aufgabe 10.8: Das Sortieren von Tabelleneinträgen unter Benutzung eines benutzerdefinierten Table-Modells	597
	☆☆☆ Aufgabe 10.9: Filtern von Tabelleneinträge mit Hilfe von Standard-Filterklassen.	597
	☆☆☆ Aufgabe 10.10: Einen benutzerdefinierten Filter erzeugen	598
	☆☆☆ Aufgabe 10.11: Wiederholungsaufgabe	599
10.7	Komponenten für die Registerkarten (Tabs) von JTabbedPane-Instanzen nutzen	599
	☆☆ Aufgabe 10.12 : JTabbedPane-Registerkarten mit einer String-Beschriftung.	600
	☆☆ Aufgabe 10.13: JTabbedPane-Registerkarten mit Komponenten	600
10.8	Drag-and-Drop-Unterstützung in Java	601
	☆ Aufgabe 10.14: Drag-and-Drop-Test für JTextArea- und JList-Komponenten	602
	☆☆ Aufgabe 10.15: Eigenschaften für den Transfer definieren	603

	☆☆	Aufgabe 10.16: Drag-and-Drop-Test für JLabel-Komponenten	604
	☆☆	Aufgabe 10.17: Die Klassen Clipboard, StringSelection und DataFlavor	604
	☆☆☆	Aufgabe 10.18: Das Interface DropTargetListener und die Klasse DropTargetEvent	605
10.9		Benutzerdefinierte Transfer-Handler	607
	☆☆☆	Aufgabe 10.19: Benutzerdefinierter Transfer-Handler für JList-Komponenten	607
	☆☆☆	Aufgabe 10.20: Benutzerdefinierter Transfer-Handler für JTree-Komponenten.....	608
10.10		Die neuen Drag-and-Drop-Klassen aus der Java-Version 6.0	609
	☆☆☆	Aufgabe 10.21: Eine neue Betrachtung der Drop-Fähigkeit für JList-Komponenten	610
	☆☆☆	Aufgabe 10.22: Eine neue Betrachtung der Drop-Fähigkeit für JTree-Komponenten	611
10.11		Lösungen	612
		Lösung 10.1	612
		Lösung 10.2	614
		Lösung 10.3	616
		Lösung 10.4	618
		Lösung 10.5	619
		Lösung 10.6	620
		Lösung 10.7	622
		Lösung 10.8	624
		Lösung 10.9	625
		Lösung 10.10	628
		Lösung 10.11	630
		Lösung 10.12	632
		Lösung 10.13	633
		Lösung 10.14	636
		Lösung 10.15	637
		Lösung 10.16	638
		Lösung 10.17	639
		Lösung 10.18	642
		Lösung 10.19	645
		Lösung 10.20	648
		Lösung 10.21	652
		Lösung 10.22	655

II	Neue Features von Java 7	659
II.1	Strings in switch-Anweisungen und neue Literale mit Java 7	659
	☆ Aufgabe 11.1: Binäre Literale, Underscores in numerischen Literalen und Strings in switch-Anweisungen	660
II.2	Typen in Java	660
II.3	Typprüfung und Typsicherheit mittels Generics	664
II.4	Subtyping für parametrisierte Typen	670
II.5	Die extends-Klausel	671
II.6	Typinferenz für Methoden	672
II.7	Typinferenz beim Erzeugen von Instanzen eines generischen Typs	673
II.8	Heap Pollution	675
II.9	Wildcard-Capture	677
	☆ Aufgabe 11.2: Typinferenz beim Instantiieren von generischen Klassen	678
	☆ Aufgabe 11.3: Der Diamond-Operator	679
	☆☆ Aufgabe 11.4: Schranken für Typvariablen und Typinferenz für Methoden	680
	☆☆☆ Aufgabe 11.5: Parametrisierte Typen und Wildcardtypen	683
	☆☆ Aufgabe 11.6: Generische Arraytypen	687
	☆☆ Aufgabe 11.7: Subtyping von Referenztypen	691
II.10	Multi-catch-Klausel und verbesserte Typprüfung beim Rethrowing von Exceptions	692
	☆ Aufgabe 11.8: Disjunction-Typ für Exceptions	693
	☆☆ Aufgabe 11.9: Typprüfung beim Rethrowing von Exceptions	695
II.11	Transparente und nicht-rechteckige Fenster mit Java 7 erzeugen ...	697
	☆☆ Aufgabe 11.10: Fensterdekorationen	698
	☆☆ Aufgabe 11.11: Transparente und nicht-rechteckige Fenster	699
II.12	Das Überlappen von LW- und HW-Komponenten mit Java 7	701
	☆☆ Aufgabe 11.12: LW- und HW-Komponenten überlappen	701
II.13	Das Nimbus-LookAndFeel	702
	☆☆ Aufgabe 11.13: Das Nimbus-LookAndFeel und das Skinning von Komponenten	704
II.14	Swing-Komponenten mit einem JLayer dekorieren	706

	☆	Aufgabe II.14: Das Dekorieren von Swing-Komponenten	707
	☆☆	Aufgabe II.15: JLayer- und LayerUI-Instanzen	708
	☆☆	Aufgabe II.16: Event-Handling für JLayer- und LayerUI-Instanzen	709
II.15		Andere Erweiterungen	711
II.16		Lösungen	713
		Lösung II.1.	713
		Lösung II.2.	716
		Lösung II.3.	720
		Lösung II.4.	725
		Lösung II.5.	731
		Lösung II.6.	748
		Lösung II.7.	754
		Lösung II.8.	761
		Lösung II.9.	763
		Lösung II.10.	767
		Lösung II.11.	769
		Lösung II.12.	775
		Lösung II.13.	779
		Lösung II.14.	784
		Lösung II.15.	786
		Lösung II.16.	789
12		Java 8 Lambdas und Streams	795
12.1		Mittels anonymer Klassen Code an Methoden übergeben	795
12.2		Funktionale Interfaces.	797
12.3		Syntax und Deklaration von Lambda-Ausdrücken	797
	☆	Aufgabe 12.1: Lambda-Ausdruck ohne Parameter versus anonymer Klasse	802
	☆	Aufgabe 12.2: Lambda-Ausdruck mit Parameter versus anonymer Klasse	805
	☆	Aufgabe 12.3: Weitere Beispiele mit anonymen Klassen und Lambda-Ausdrücken.	805
12.4		Scoping und Variable Capture	806
	☆☆	Aufgabe 12.4: Die Umgebung von Lambda-Ausdrücken.	807
	☆☆	Aufgabe 12.5: Die neuen funktionalen Interfaces Consumer<T> und Predicate<T> und die Übergabe von Lambda-Ausdrücken in Methoden.	808

	☆☆	Aufgabe 12.6: Wiederholungsaufgabe	810
12.5		Methoden- und Konstruktor-Referenzen	813
	☆	Aufgabe 12.7: Methoden-Referenzen in Zuweisungen	815
	☆☆	Aufgabe 12.8: Methoden-Referenzen als Argumente in Methodenaufrufen übergeben	816
	☆☆	Aufgabe 12.9: Konstruktor-Referenzen und die neuen funktionalen Interfaces Supplier<T> und Function<T,R>	818
12.6		Default-Methoden und statische Methoden in Interfaces	820
12.7		Das neue Interface Stream	822
12.8		Die forEach-Methoden von Iterator, Iterable und Stream	826
12.9		Die Default-Methoden des Map-Interface	828
	☆	Aufgabe 12.10: Die forEach()-Methode von Iterable ...	830
	☆	Aufgabe 12.11: Die forEach()-Methode des Iterator-Interface	831
	☆	Aufgabe 12.12: Die funktionalen Interfaces BiConsumer<T,U>, BiPredicate<T,U> und BiFunction<T,U,R>	832
	☆☆	Aufgabe 12.13: Die Methoden des Interface Stream und die Behandlung von Exceptions in Lambda-Ausdrücken	834
	☆	Aufgabe 12.14: Die forEach- und replace-Methoden des Map-Interface	837
12.10		Das Interface Collector und die Klasse Collectors. Reduktion mittels Methoden von Streams und Kollektoren	838
	☆☆	Aufgabe 12.15: Weitere Methoden des Interface Stream: limit(), count(), max(), min(), skip(), reduce() und collect()	843
	☆☆☆	Aufgabe 12.16: Das Interface Collector und die Klasse Collectors	850
12.11		Parallele Streams	852
12.12		Die Bulk-Operationen der Klasse ConcurrentHashMap	856
	☆☆☆	Aufgabe 12.17: Parallele Streams	859
	☆☆☆	Aufgabe 12.18 : Die neuen Methoden von ConcurrentHashMap	861
12.13		Lösungen	863
		Lösung 12.1	863
		Lösung 12.2	870
		Lösung 12.3	872

Lösung 12.4	874
Lösung 12.5	879
Lösung 12.6	883
Lösung 12.7	892
Lösung 12.8	893
Lösung 12.9	903
Lösung 12.10	908
Lösung 12.11	909
Lösung 12.12	910
Lösung 12.13	914
Lösung 12.14	924
Lösung 12.15	928
Lösung 12.16	949
Lösung 12.17	957
Lösung 12.18	972
Stichwortverzeichnis	981