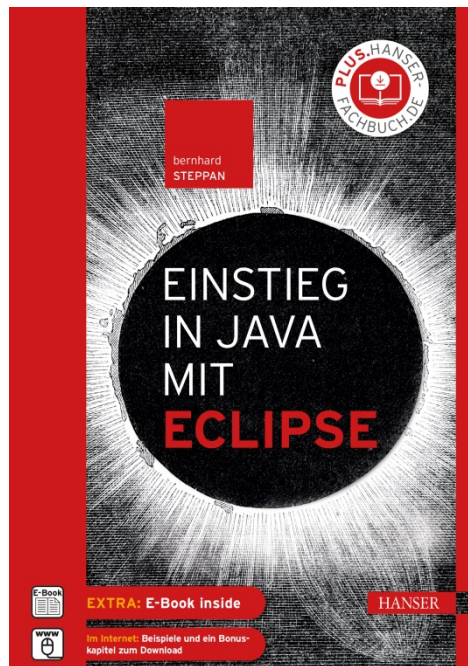


# HANSER



## Leseprobe

zu

## „Einstieg in Java mit Eclipse“

von Bernhard Steppan

Print-ISBN: 978-3-446-45910-6  
E-Book-ISBN: 978-3-446-45976-2  
E-Pub-ISBN: 978-3-446-46326-4

Weitere Informationen und Bestellungen unter  
<http://www.hanser-fachbuch.de/978-3-446-45910-6>

sowie im Buchhandel

© Carl Hanser Verlag, München

# Inhalt

<b>Vorwort .....</b>	<b>XXV</b>
<b>Teil I: Grundlagen .....</b>	<b>1</b>
<b>1 Programmiergrundlagen.....</b>	<b>3</b>
1.1 Einleitung .....	3
1.2 Die Sprache der Maschinenwelt .....	4
1.3 Hochsprache als Kompromiss .....	6
1.4 Entwicklungsumgebung .....	7
1.4.1 Compiler.....	7
1.4.2 Editor.....	7
1.4.3 Projektverwaltung.....	7
1.5 Laufzeitumgebung.....	8
1.6 Zusammenfassung .....	8
1.7 Aufgaben .....	9
1.8 Literatur .....	9
<b>2 Technologieüberblick .....</b>	<b>11</b>
2.1 Einleitung .....	11
2.2 Überblick.....	12
2.2.1 Die Anfangszeit von Java .....	12
2.2.2 Die Reifezeit von Java .....	13
2.2.3 Die Gegenwart von Java .....	14
2.3 Warum Java? .....	15
2.3.1 Leicht lesbar.....	15
2.3.2 Objektorientiert.....	15
2.3.3 Sicher und robust.....	15

2.3.4	Leistungsfähig .....	16
2.3.5	Universell verwendbar .....	16
2.3.6	Kostenfrei .....	16
2.3.7	Quelloffen .....	16
2.3.8	Leicht portierbar .....	16
2.3.9	Java-Programme lassen sich leicht erweitern .....	17
2.3.10	Java-Programme lassen sich leicht entwickeln und testen .....	17
2.4	Was gehört zu Java? .....	18
2.4.1	Sprache Java .....	18
2.4.2	Java Virtual Machine .....	18
2.4.3	Klassenbibliotheken .....	20
2.4.4	Java-Werkzeuge .....	20
2.5	Java-Versionen .....	21
2.6	Java-Editionen .....	21
2.6.1	Java Standard Edition .....	21
2.6.2	Java Enterprise Edition .....	21
2.6.3	Java Micro Edition .....	21
2.7	Zusammenfassung .....	22
2.8	Aufgaben .....	23
2.9	Literatur .....	23
<b>3</b>	<b>Objektorientierte Programmierung .....</b>	<b>25</b>
3.1	Einleitung .....	25
3.2	Überblick .....	26
3.3	Objekt .....	27
3.4	Klasse .....	28
3.4.1	Attribute .....	28
3.4.2	Methoden .....	30
3.5	Abstraktion .....	32
3.6	Vererbung .....	33
3.6.1	Basisklassen .....	34
3.6.2	Abgeleitete Klassen .....	35
3.6.3	Mehrfachvererbung .....	36
3.7	Sichtbarkeit .....	37
3.8	Beziehungen .....	39
3.8.1	Beziehungen ohne Vererbung .....	39
3.8.2	Vererbungsbeziehungen .....	41
3.9	Designfehler .....	43

3.10	Umstrukturierung .....	44
3.11	Modellierung .....	44
3.12	Persistenz .....	44
3.13	Polymorphie .....	44
3.13.1	Statische Polymorphie .....	45
3.13.2	Dynamische Polymorphie .....	45
3.14	Designregeln .....	46
3.15	Zusammenfassung .....	47
3.16	Aufgaben .....	48
3.17	Literatur .....	48
<b>4</b>	<b>Entwicklungsumgebung .....</b>	<b>49</b>
4.1	Einleitung .....	49
4.2	Installation .....	50
4.2.1	Betriebssystem .....	50
4.2.2	Java installieren .....	51
4.2.3	Eclipse installieren .....	54
4.2.4	Beispielprogramme installieren .....	60
4.2.5	Installation überprüfen .....	64
4.3	Einführung in Eclipse .....	67
4.3.1	Überblick .....	67
4.3.2	Workbench .....	68
4.3.3	Perspektiven, Sichten und Editoren .....	68
4.3.4	Package Explorer .....	71
4.3.5	Java-Editor .....	72
4.3.6	Code-Formatierer .....	75
4.3.7	Build-System .....	77
4.3.8	Debugger .....	78
4.3.9	Modularer Aufbau .....	78
4.3.10	Eclipse-Workspace .....	79
4.3.11	Softwareaktualisierung .....	80
4.3.12	Hilfesystem .....	81
4.4	Zusammenfassung .....	82
4.5	Aufgaben .....	83
4.6	Literatur .....	83

<b>Teil II: Sprache Java</b> .....	<b>85</b>
<b>5 Programmaufbau</b> .....	<b>87</b>
5.1 Einleitung .....	87
5.2 Überblick.....	88
5.3 Sprachelemente des Programms .....	89
5.3.1 Kommentar.....	90
5.3.2 Pakete .....	90
5.3.3 Klassen .....	91
5.3.4 Methoden.....	92
5.3.5 Anweisungen.....	93
5.4 Struktur des Programms .....	95
5.5 Ablauf des Programms .....	95
5.6 Reservierte Schlüsselwörter .....	96
5.7 Zusammenfassung .....	98
5.8 Übungen .....	99
5.8.1 Eclipse starten .....	99
5.8.2 Workspace »Uebungen« auswählen .....	99
5.8.3 Dialog »New Java Project« aufrufen .....	100
5.8.4 Module abwählen und ein neues Projekt erzeugen .....	100
5.8.5 Dialog »New Java Class« aufrufen .....	101
5.8.6 Klasse »Person« erzeugen .....	101
5.8.7 Entwicklungsumgebung einrichten .....	102
5.8.8 Perspektive speichern.....	104
5.8.9 Attribut einfügen .....	105
5.8.10 Konstruktor erzeugen.....	106
5.8.11 Getter und Setter erzeugen.....	108
5.8.12 Klasse »Programmdemo« erzeugen.....	112
5.8.13 Klasse »Programmdemo« komplettieren.....	113
5.8.14 Programm starten .....	114
5.9 Aufgaben .....	114
<b>6 Variablen</b> .....	<b>115</b>
6.1 Einleitung .....	115
6.2 Überblick.....	116
6.2.1 Zweck von Variablen .....	116
6.2.2 Arten von Variablen .....	116
6.2.3 Verwendung von Variablen .....	117
6.3 Lokale Variablen .....	119

6.4	Parameter .....	120
6.5	Objektvariablen .....	121
6.5.1	Individuelle Objektvariablen .....	121
6.5.2	Objektvariable »this« .....	122
6.6	Klassenvariablen .....	124
6.7	Konstanten .....	126
6.8	Zusammenfassung .....	128
6.9	Übungen .....	129
6.9.1	Eclipse starten .....	129
6.9.2	Projekt kopieren .....	129
6.9.3	Attribut einfügen .....	130
6.9.4	Attribut umbenennen .....	131
6.9.5	Konstruktor anpassen .....	134
6.9.6	Klasse »Programmdemo« anpassen .....	135
6.9.7	Programm starten .....	136
6.10	Aufgaben .....	137
6.11	Literatur .....	137
<b>7</b>	<b>Anweisungen .....</b>	<b>139</b>
7.1	Einleitung .....	139
7.2	Überblick .....	140
7.2.1	Zweck von Anweisungen .....	140
7.2.2	Arten von Anweisungen .....	141
7.3	Deklaration .....	141
7.4	Zuweisung .....	143
7.4.1	Aufbau der Java-Zuweisung .....	143
7.4.2	Java-Zuweisung ungleich mathematische Gleichungen .....	143
7.4.3	Ist $x = y$ gleich $y = x$ ? .....	144
7.4.4	Kombination aus Deklaration und Wertzuweisung .....	146
7.5	Block .....	146
7.6	Variablenaufruf .....	150
7.7	Methodenaufruf .....	151
7.8	Zusammenfassung .....	152
7.9	Übungen .....	153
7.9.1	Eclipse starten .....	153
7.9.2	Projekt kopieren .....	153
7.9.3	Anweisung einfügen .....	154
7.9.4	Klasse »Programmdemo« anpassen .....	156
7.9.5	Programm starten .....	157

7.10	Aufgaben .....	157
7.11	Literatur .....	158
<b>8</b>	<b>Einfache Datentypen .....</b>	<b>159</b>
8.1	Einleitung .....	159
8.2	Überblick.....	160
8.2.1	Zweck von einfachen Datentypen.....	160
8.2.2	Arten von einfachen Datentypen .....	160
8.2.3	Verwendung von einfachen Datentypen .....	160
8.3	Ganzzahlen .....	164
8.3.1	Datentyp »byte«.....	164
8.3.2	Datentyp »short« .....	165
8.3.3	Datentyp »int« .....	166
8.3.4	Datentyp »long« .....	167
8.4	Kommazahlen .....	168
8.4.1	Datentyp »float« .....	168
8.4.2	Datentyp »double« .....	169
8.5	Zeichen .....	170
8.6	Wahrheitswerte .....	170
8.7	Zusammenfassung .....	171
8.8	Übungen .....	172
8.8.1	Eclipse starten .....	173
8.8.2	Projekt kopieren .....	173
8.8.3	Attribute einfügen .....	174
8.8.4	Dateien im Editor schließen .....	174
8.8.5	Konstruktor anpassen .....	175
8.8.6	Konstruktoraufruf anpassen .....	176
8.8.7	Abfragemethode erzeugen .....	177
8.8.8	Programmausgabe anpassen .....	178
8.8.9	Programm starten .....	179
8.8.10	Startkonfigurationen aufräumen.....	179
8.9	Aufgaben .....	181
8.10	Literatur .....	181
<b>9</b>	<b>Klassen und Objekte .....</b>	<b>183</b>
9.1	Einleitung .....	183
9.2	Überblick.....	184
9.2.1	Zweck einer Klasse .....	184
9.2.2	Arten von Klassen .....	185

9.2.3	Definition von Klassen.....	185
9.2.4	Verwendung von Klassen.....	186
9.3	Konkrete Klasse .....	189
9.3.1	Konkrete Klasse definieren.....	189
9.3.2	Objekte einer konkreten Klasse erzeugen.....	190
9.3.3	Innere Klasse .....	192
9.3.4	Lokale Klasse .....	194
9.3.5	Anonyme Klasse .....	195
9.3.6	Vererbung.....	197
9.4	Abstrakte Klassen .....	202
9.5	Interfaces.....	204
9.6	Generics.....	207
9.6.1	Generische Klasse definieren .....	207
9.6.2	Objekte erzeugen .....	208
9.7	Zusammenfassung .....	212
9.8	Übungen .....	213
9.8.1	Eclipse starten .....	213
9.8.2	Projekt kopieren .....	213
9.8.3	Dateien schließen .....	214
9.8.4	Neue Klasse extrahieren .....	214
9.8.5	Methode »getName()« verändern .....	217
9.8.6	Attribut »name« anlegen .....	218
9.8.7	Klasse »Person« umbenennen .....	219
9.8.8	Startkonfiguration anpassen.....	220
9.8.9	Hauptprogramm »Programmdemo« ausführen.....	221
9.8.10	Konstruktor »Wesen« anpassen .....	222
9.8.11	Konstruktor »Mensch« anpassen.....	222
9.8.12	Programm »Programmdemo« erneut starten .....	223
9.9	Aufgaben .....	223
9.10	Literatur .....	224
<b>10</b>	<b>Aufzählungen .....</b>	<b>225</b>
10.1	Einleitung .....	225
10.2	Überblick.....	226
10.2.1	Zweck von Enums .....	226
10.2.2	Definition und Deklaration von Enums .....	228
10.2.3	Verwendung von Enums .....	229



10.3	Aufzählungsklassen.....	230
10.3.1	Konstruktor.....	230
10.3.2	Methode »value()«.....	231
10.3.3	Eigenständige einfache Enum-Klasse .....	231
10.3.4	Eigenständige erweiterte Enum-Klasse .....	232
10.3.5	Innere erweiterte Enum-Klasse .....	234
10.4	Zusammenfassung .....	235
10.5	Übungen .....	236
10.5.1	Eclipse starten .....	236
10.5.2	Projekt kopieren .....	236
10.5.3	Dateien schließen .....	237
10.5.4	Dialog »New Enum« aufrufen.....	237
10.5.5	Enum-Klasse »Anrede« erzeugen.....	238
10.5.6	Elemente einfügen .....	239
10.5.7	Konstruktor erzeugen.....	239
10.5.8	Tasks anzeigen .....	239
10.5.9	Konstruktor erweitern .....	240
10.5.10	Attribut »name« als Objektvariable einfügen .....	241
10.5.11	Methode »toString()« überschreiben .....	242
10.5.12	Methode »toString()« erweitern .....	243
10.5.13	Attribut »anrede« hinzufügen.....	243
10.5.14	Konstruktor der Klasse »Wesen« erweitern .....	243
10.5.15	Abfragemethode »getAnrede()« erzeugen.....	244
10.5.16	Konstruktor der Klasse »Mensch« erweitern.....	245
10.5.17	Klasse »Programmdemo« erweitern .....	245
10.5.18	Programm starten .....	246
10.6	Aufgaben .....	246
10.7	Literatur .....	246
<b>11</b>	<b>Arrays.....</b>	<b>247</b>
11.1	Einleitung .....	247
11.2	Überblick.....	248
11.2.1	Zweck von Arrays .....	248
11.2.2	Arten von Arrays .....	248
11.2.3	Verwendung von Arrays .....	249
11.3	Zusammenfassung .....	253
11.4	Übungen .....	254
11.4.1	Eclipse starten .....	254

11.4.2	Projekt kopieren .....	254
11.4.3	Dateien schließen .....	255
11.4.4	Klasse »Roboter« erzeugen .....	256
11.4.5	Klasse »Programmdemo« erweitern .....	257
11.4.6	Vorüberlegungen zum Aufbau des Arrays.....	257
11.4.7	Deklaration und Zuweisung .....	258
11.4.8	Array-Objekte erzeugen .....	258
11.4.9	Fehler mit Suchen und Ersetzen beseitigen .....	259
11.4.10	Cast einfügen .....	260
11.4.11	Weitere Teammitglieder hinzufügen.....	261
11.4.12	Programm starten .....	261
11.5	Aufgaben .....	262
<b>12</b>	<b>Methoden .....</b>	<b>263</b>
12.1	Einleitung .....	263
12.2	Überblick.....	264
12.2.1	Zweck von Methoden .....	264
12.2.2	Arten von Methoden .....	265
12.2.3	Definition von Methoden .....	266
12.2.4	Verwendung von Methoden .....	270
12.3	Konstruktoren .....	272
12.3.1	Standardkonstruktoren.....	272
12.3.2	Konstruktoren ohne Parameter.....	273
12.3.3	Konstruktoren mit Parametern .....	275
12.4	Destruktoren .....	277
12.5	Operationen .....	278
12.6	Abfragemethoden.....	280
12.6.1	Definition.....	280
12.6.2	Verwendung .....	282
12.7	Änderungsmethoden.....	283
12.7.1	Definition.....	283
12.7.2	Verwendung.....	284
12.8	Zusammenfassung .....	285
12.9	Übungen .....	286
12.9.1	Eclipse starten .....	286
12.9.2	Projekt kopieren .....	286
12.9.3	Dateien schließen .....	287
12.9.4	Methoden der Klasse »Wesen« erzeugen .....	288

12.9.5	Quellcode der Klasse »Wesen« kontrollieren .....	289
12.9.6	Sortierung der Klasse gegebenenfalls ändern .....	290
12.9.7	Konstruktor der Klasse »Mensch« erzeugen .....	291
12.9.8	Quellcode der Klasse »Mensch« vergleichen .....	292
12.9.9	Konstruktor in »Programmdemo« tauschen .....	293
12.9.10	Programm starten .....	294
12.10	Aufgaben .....	295
12.11	Literatur .....	295
<b>13</b>	<b>Operatoren .....</b>	<b>297</b>
13.1	Einleitung .....	297
13.2	Überblick .....	298
13.3	Arithmetische Operatoren .....	298
13.3.1	Positives Vorzeichen .....	299
13.3.2	Negatives Vorzeichen .....	299
13.3.3	Additionsoperator .....	300
13.3.4	Differenzoperator .....	301
13.3.5	Divisionsoperator .....	302
13.3.6	Modulo-Operator .....	303
13.3.7	Präinkrement-Operator .....	303
13.4	Vergleichende Operatoren .....	306
13.5	Logische Operatoren .....	310
13.5.1	Nicht-Operator .....	310
13.5.2	Und-Operator .....	310
13.6	Bitweise Operatoren .....	312
13.7	Zuweisungsoperatoren .....	312
13.8	Fragezeichenoperator .....	313
13.9	New-Operator .....	314
13.10	Cast-Operator .....	315
13.11	Zugriffsoperatoren .....	317
13.11.1	Punktoperator .....	317
13.11.2	Lambda-Operator .....	318
13.12	Zusammenfassung .....	319
13.13	Übungen .....	320
13.13.1	Eclipse starten .....	320
13.13.2	Projekt kopieren .....	320
13.13.3	Dateien schließen .....	321
13.13.4	Methode »ermittleStudentenstatus()« erzeugen .....	321

13.13.5	Methode »main()« verändern .....	322
13.13.6	Konstruktor der Klasse »Mensch« erzeugen .....	322
13.13.7	Konstruktor »Mensch« implementieren .....	323
13.13.8	Klasse »Programmdemo« kontrollieren .....	324
13.13.9	Programm starten .....	325
13.13.10	Startkonfiguration anpassen .....	325
13.14	Aufgaben .....	325
13.15	Literatur .....	325
<b>14</b>	<b>Verzweigungen .....</b>	<b>327</b>
14.1	Einleitung .....	327
14.2	Überblick .....	328
14.3	If-Verzweigung .....	328
14.4	Fragezeichenoperator-Verzweigung .....	329
14.5	Switch-Verzweigung .....	331
14.5.1	Beschränkung auf wenige Datentypen .....	331
14.5.2	Auswahl über Strings .....	332
14.5.3	Yield-Anweisung .....	333
14.5.4	Lambda-Operator .....	334
14.6	Zusammenfassung .....	335
14.7	Übungen .....	336
14.7.1	Eclipse starten .....	336
14.7.2	Projekt kopieren .....	336
14.7.3	Dateien schließen .....	337
14.7.4	Klasse »Anrede« verändern .....	337
14.7.5	Klasse »Anrede« kopieren .....	337
14.7.6	Fehlerhafte Klasse »Programmdemo« aufrufen .....	338
14.7.7	Aufzählungstyp »Geschlecht« einfügen .....	338
14.7.8	Konstruktoren der Klasse »Mensch« verändern .....	339
14.7.9	Klasse »Wesen« anpassen .....	340
14.7.10	Klasse »Wesen« kontrollieren .....	341
14.7.11	Klasse »Roboter« anpassen .....	342
14.7.12	Klasse »Mensch« anpassen .....	342
14.7.13	Klasse »Programmdemo« kontrollieren .....	344
14.7.14	Methode »ermittleAnrede()« einfügen .....	344
14.7.15	Methode »main()« ändern .....	345
14.7.16	Programm starten .....	346
14.7.17	Startkonfiguration anpassen .....	346
14.8	Aufgaben .....	347
14.9	Literatur .....	347

<b>15 Schleifen</b> .....	<b>349</b>
15.1 Einleitung .....	349
15.2 Überblick .....	350
15.2.1 Zweck von Schleifen .....	350
15.2.2 Arten von Schleifen .....	350
15.3 While-Schleife .....	351
15.4 Do-Schleife .....	352
15.5 Einfache For-Schleife .....	353
15.6 Erweiterte For-Schleife .....	354
15.7 Zusammenfassung .....	355
15.8 Übungen .....	356
15.8.1 Vorüberlegungen .....	356
15.8.2 Eclipse starten .....	356
15.8.3 Projekt kopieren .....	356
15.8.4 Dateien im Editor schließen .....	357
15.8.5 Erzeugen der Personenobjekte ergänzen .....	357
15.8.6 Redundanzen erkennen .....	357
15.8.7 Schleife erzeugen .....	358
15.8.8 Index in den Schleifenkörper übertragen .....	358
15.8.9 Schleife optimieren .....	359
15.8.10 Programm starten .....	360
15.8.11 Startkonfiguration anpassen .....	360
15.9 Aufgaben .....	361
15.10 Literatur .....	361
<b>16 Pakete und Module</b> .....	<b>363</b>
16.1 Einleitung .....	363
16.2 Überblick .....	364
16.3 Pakete .....	364
16.3.1 Klassenimport .....	364
16.3.2 Namensräume .....	366
16.4 Module .....	368
16.5 Zusammenfassung .....	370
16.6 Übungen .....	371
16.6.1 Eclipse starten .....	371
16.6.2 Projekt kopieren .....	371
16.6.3 Dateien im Editor schließen .....	371
16.6.4 Dialog »Rename Package« aufrufen .....	372

16.6.5	Paket »programmierkurs« umbenennen .....	372
16.6.6	Auswirkungen des Refactorings kontrollieren .....	373
16.6.7	Dialog »New Package« aufrufen .....	373
16.6.8	Neues Paket erzeugen.....	374
16.6.9	Restliche Klassen in ein neues Paket verschieben .....	376
16.6.10	Refactoring im »Package Explorer« kontrollieren .....	376
16.6.11	Änderungen an der Klasse »Programmdemo« kontrollieren .....	376
16.6.12	Programm starten .....	377
16.6.13	Startkonfiguration anpassen.....	378
16.7	Aufgaben .....	378
<b>17</b>	<b>Fehlerbehandlung .....</b>	<b>379</b>
17.1	Einleitung .....	379
17.2	Überblick.....	380
17.2.1	Motivation.....	380
17.2.2	Arten von Fehlern .....	380
17.2.3	Verwendung des Exception Handlings .....	381
17.3	Basisklasse »Throwable« .....	384
17.4	Klasse »Error«.....	385
17.4.1	Subklasse »OutOfMemoryError« .....	385
17.4.2	Subklasse »StackOverflowError« .....	387
17.5	Klasse »Exception« .....	388
17.5.1	Subklasse »RuntimeException«.....	389
17.5.2	Subklasse »IOException« .....	389
17.5.3	Eigene Exceptions .....	390
17.6	Zusammenfassung .....	393
17.7	Übungen .....	394
17.7.1	Vorüberlegungen .....	394
17.7.2	Eclipse starten .....	394
17.7.3	Projekt kopieren .....	394
17.7.4	Dateien im Editor schließen .....	394
17.7.5	Neue Java-Klasse anlegen .....	395
17.7.6	Klasse »WahlpflichtfachNichtBelegtException« kontrollieren .....	395
17.7.7	Klasse »Person« erweitern .....	396
17.7.8	Konstruktor auf fünf Parameter erweitern .....	397
17.7.9	Getter-Methode erzeugen .....	397
17.7.10	Klasse »Programmdemo« erweitern .....	398
17.7.11	Programm starten .....	399
17.7.12	Startkonfiguration anpassen.....	400

17.8	Aufgaben .....	400
17.9	Literatur .....	400
<b>18</b>	<b>Dokumentation .....</b>	<b>401</b>
18.1	Einleitung .....	401
18.2	Überblick.....	402
18.3	Zeilenkommentare .....	402
18.4	Blockkommentare .....	403
18.5	Dokumentationskommentare .....	403
18.6	Zusammenfassung .....	405
18.7	Übungen .....	406
18.7.1	Eclipse starten .....	406
18.7.2	Projekt kopieren .....	406
18.7.3	Dateien im Editor schließen .....	406
18.7.4	Javadoc erzeugen .....	406
18.7.5	Autorenname .....	407
18.7.6	Javadoc in Eclipse anzeigen .....	407
18.7.7	Blockkommentar erzeugen .....	408
18.7.8	Zeilenbezogenen Kommentar erzeugen .....	408
18.7.9	Tasks.....	408
18.8	Aufgaben .....	409
18.9	Literatur .....	409
<b>Teil III:</b>	<b>Plattform Java .....</b>	<b>411</b>
<b>19</b>	<b>Entwicklungsprozesse .....</b>	<b>413</b>
19.1	Einleitung .....	413
19.2	Überblick.....	414
19.2.1	Zusammenhang zwischen Phasen und Aktivitäten .....	415
19.2.2	Aktivitäten .....	416
19.2.3	Werkzeuge .....	417
19.3	Planungsphase .....	417
19.3.1	Auftragsklärung.....	417
19.3.2	Anforderungsaufnahme .....	418
19.4	Konstruktionsphase .....	418
19.4.1	Analyse .....	418
19.4.2	Design.....	419

19.4.3	Implementierung .....	420
19.4.4	Test .....	432
19.5	Betriebsphase .....	437
19.5.1	Verteilung .....	437
19.5.2	Pflege .....	439
19.6	Zusammenfassung .....	440
19.7	Aufgaben .....	440
19.8	Literatur .....	440
<b>20</b>	<b>Laufzeitumgebung .....</b>	<b>441</b>
20.1	Einleitung .....	441
20.2	Überblick .....	442
20.3	Bytecode .....	443
20.4	Java Virtual Machine .....	447
20.4.1	Künstlicher Computer .....	447
20.4.2	Interpreter-Modus .....	448
20.4.3	JIT-Compiler-Modus .....	448
20.4.4	Hotspot-Modus .....	449
20.4.5	Garbage Collector .....	449
20.5	Bibliotheken .....	450
20.5.1	Native Bibliotheken .....	450
20.5.2	Klassenbibliotheken .....	450
20.5.3	Ressourcen und Property-Dateien .....	451
20.6	Portabilität eines Java-Programms .....	451
20.6.1	Binärkompatibler Bytecode .....	451
20.6.2	Voraussetzungen beim Portieren .....	452
20.7	Programmstart .....	452
20.7.1	Startskript .....	452
20.7.2	Nativer Wrapper .....	453
20.8	JVM-Konfiguration .....	455
20.9	Zusammenfassung .....	456
20.10	Aufgaben .....	457
20.11	Literatur .....	457
<b>21</b>	<b>Klassenbibliotheken .....</b>	<b>459</b>
21.1	Einleitung .....	459
21.2	Überblick .....	460
21.2.1	Einsatzbereiche .....	461
21.2.2	Wiederverwendung .....	461



21.2.3	Dokumentation .....	461
21.2.4	Spracherweiterung .....	461
21.2.5	Arten von Klassenbibliotheken .....	461
21.3	Java Standard Edition .....	462
21.3.1	Basisklassen .....	463
21.3.2	Klasse »System« .....	470
21.3.3	Threads .....	473
21.3.4	Streams .....	475
21.3.5	Properties .....	477
21.3.6	Container-Klassen .....	479
21.3.7	Abstract Windowing Toolkit .....	480
21.3.8	Swing .....	489
21.3.9	JavaBeans .....	493
21.3.10	Applets .....	493
21.3.11	Java Database Connectivity (JDBC) .....	493
21.3.12	Java Native Interface .....	494
21.3.13	Remote Method Invocation .....	495
21.4	Java Enterprise Edition .....	496
21.4.1	Entity Beans .....	497
21.4.2	Session Beans .....	497
21.4.3	Message Driven Beans .....	497
21.4.4	Schnittstellen .....	498
21.5	Java Micro Edition .....	498
21.6	Externe Klassenbibliotheken .....	499
21.6.1	Apache Software Foundation .....	499
21.6.2	Eclipse Community .....	499
21.6.3	SourceForge .....	500
21.6.4	Weitere quelloffene Software .....	500
21.6.5	Kommerzielle Software .....	500
21.7	Zusammenfassung .....	500
21.8	Aufgaben .....	501
21.9	Literatur .....	501
<b>22</b>	<b>Gesetzmäßigkeiten .....</b>	<b>503</b>
22.1	Einleitung .....	503
22.2	Überblick .....	504
22.3	Schreibweisen .....	505

22.4	Sichtbarkeit .....	506
22.4.1	Vier Sichtbarkeitsbereiche .....	506
22.4.2	Sichtbarkeit »private« .....	507
22.4.3	Sichtbarkeit »default« .....	507
22.4.4	Sichtbarkeit »protected« .....	507
22.4.5	Sichtbarkeit »public« .....	507
22.4.6	Fallbeispiel .....	508
22.4.7	Gültigkeitsbereich von Variablen .....	513
22.5	Auswertungsreihenfolge .....	518
22.5.1	Punkt vor Strich .....	518
22.5.2	Punkt vor Punkt .....	519
22.6	Typkonvertierung .....	521
22.6.1	Implizite Konvertierung .....	522
22.6.2	Explizite Konvertierung .....	523
22.7	Polymorphie .....	526
22.7.1	Überladen von Methoden .....	526
22.7.2	Überschreiben von Methoden .....	528
22.8	Zusammenfassung .....	532
22.9	Aufgaben .....	533
22.10	Literatur .....	533
<b>23</b>	<b>Algorithmen .....</b>	<b>535</b>
23.1	Einleitung .....	535
23.2	Überblick .....	536
23.2.1	Algorithmen entwickeln .....	536
23.2.2	Algorithmenarten .....	537
23.2.3	Algorithmen verwenden .....	537
23.3	Algorithmen entwickeln .....	538
23.3.1	Sortieralgorithmen .....	538
23.3.2	Grafikalgorithmen .....	539
23.4	Algorithmen verwenden .....	547
23.4.1	Sortieralgorithmen .....	547
23.4.2	Suchalgorithmen .....	548
23.5	Zusammenfassung .....	550
23.6	Aufgaben .....	551
23.7	Literatur .....	551

<b>Teil IV: Java-Projekte</b> .....	<b>553</b>
<b>24 Swing-Programme</b> .....	<b>555</b>
24.1 Einleitung .....	555
24.2 Anforderungen .....	556
24.3 Analyse und Design .....	556
24.3.1 Programmaufbau .....	557
24.3.2 Programmfunktionen .....	557
24.4 Implementierung .....	561
24.4.1 Eclipse mit dem Workspace »Übungen« starten .....	561
24.4.2 Neues Java-Projekt »Swing-Programme« erzeugen .....	561
24.4.3 Neue Klasse »KursstatistikApp« erzeugen .....	561
24.4.4 Klasse »KursstatistikApp« implementieren .....	561
24.4.5 Neue Klasse »Hauptfenster« erzeugen .....	562
24.4.6 Klasse »Hauptfenster« implementieren .....	563
24.4.7 Klasse »CsvParser« implementieren .....	574
24.4.8 Klasse »Tabellenfilter« implementieren .....	576
24.5 Test .....	578
24.6 Verteilung .....	579
24.7 Zusammenfassung .....	581
<b>Teil V: Anhang</b> .....	<b>583</b>
<b>25 Lösungen</b> .....	<b>585</b>
zu Kapitel 1: Programmiergrundlagen .....	585
zu Kapitel 2: Technologieüberblick .....	586
zu Kapitel 3: Objektorientierte Programmierung .....	586
zu Kapitel 4: Entwicklungsumgebung .....	588
zu Kapitel 5: Programmaufbau .....	589
zu Kapitel 6: Variablen .....	590
zu Kapitel 7: Anweisungen .....	591
zu Kapitel 8: Einfache Datentypen .....	592
zu Kapitel 9: Klassen und Objekte .....	593
zu Kapitel 10: Aufzählungen .....	595
zu Kapitel 11: Arrays .....	596
zu Kapitel 12: Methoden .....	598
zu Kapitel 13: Operatoren .....	600
zu Kapitel 14: Verzweigungen .....	601

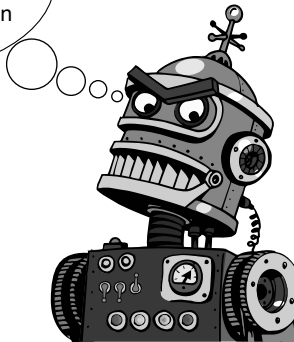
zu Kapitel 15: Schleifen .....	602
zu Kapitel 16: Pakete und Module .....	603
zu Kapitel 17: Fehlerbehandlung .....	603
zu Kapitel 18: Dokumentation .....	604
zu Kapitel 19: Entwicklungsprozesse .....	604
zu Kapitel 20: Laufzeitumgebung .....	605
zu Kapitel 21: Klassenbibliotheken .....	605
zu Kapitel 22: Gesetzmäßigkeiten .....	606
zu Kapitel 23: Algorithmen .....	606
<b>26 Bits und Bytes .....</b>	<b>609</b>
26.1 Einleitung .....	609
26.2 Zahlensysteme .....	609
26.2.1 Dezimalsystem .....	609
26.2.2 Binärsystem .....	610
26.2.3 Hexadezimalsystem .....	612
26.3 Informationseinheiten .....	613
26.3.1 Bit .....	613
26.3.2 Byte .....	613
26.3.3 Wort .....	613
26.4 Kodierung von Zeichen .....	614
26.5 Kodierung logischer Informationen .....	615
26.5.1 Und-Funktion .....	615
26.5.2 Oder-Funktion .....	616
26.5.3 Nicht-Funktion .....	616
26.6 Zusammenfassung .....	617
<b>27 Häufige Fehler .....</b>	<b>619</b>
27.1 Einleitung .....	619
27.2 Java-Fehler .....	619
27.2.1 Cannot make a static reference to the non-static field .....	619
27.2.2 Ausgabe des Werts »null« .....	620
27.2.3 NullPointerException .....	621
27.2.4 Fehlendes Break in Case-Anweisung .....	623
27.2.5 Fehlerhafter Vergleich .....	624
27.2.6 Exception ignorieren .....	627
27.2.7 NoClassDefFoundError .....	628
27.2.8 ClassNotFoundException .....	628

27.3	Eclipse-Fehler .....	628
27.3.1	Eclipse konnte nicht gestartet werden .....	628
27.3.2	Chaotische Perspektive .....	629
27.3.3	Fehlendes Fenster .....	629
27.4	Zusammenfassung .....	629
27.5	Literatur .....	629
<b>28</b>	<b>Glossar .....</b>	<b>631</b>
	<b>Stichwortverzeichnis .....</b>	<b>633</b>

# Vorwort

Java ist zurzeit unumstritten die bedeutendste Programmiersprache. Daher möchten viele Java lernen. Der Einstieg ist leider nicht einfach, denn für das Programmieren mit Java sind mindestens zwei Dinge erforderlich: das Beherrschen der Programmiersprache *und* das Beherrschen einer Entwicklungsumgebung. Aus diesem Grund ist dieses Buch entstanden. Es zeigt anhand vieler Beispiele, wie die Sprache aufgebaut ist. Zusätzlich vermittelt das Buch am Beispiel der Eclipse-Entwicklungsumgebung, wie Sie mit diesem Werkzeug Java-Programme entwickeln.

Ach herrje,  
fünf Teile mit 28 Kapiteln  
und über 600 Seiten –  
zum Schluss denken die  
Leser noch, sie könnten  
mich programmieren!



Robert aus der Maschinenwelt begleitet Sie durch das Buch.

Der erste Teil »Grundlagen« vermittelt das Java- und Eclipse-Basiswissen. Dieser Teil legt die Programmiergrundlagen, gibt Ihnen einen Überblick über die Java-Technologie und zeigt Ihnen, was das Besondere an objektorientierter Programmierung ist. Ein Kapitel über die Eclipse-Entwicklungsumgebung rundet diesen Teil ab.

Im zweiten Teil »Sprache Java« dreht sich alles um die Feinheiten der Sprache Java. Hier entstehen die ersten kleinen Java-Anwendungen. Dieser Teil bietet eine Mischung aus Wissensteil und praktischen Übungen. An jedem Kapitelende finden Sie Aufgaben, die Sie selbstständig durchführen können. Mit den Lösungen zu den Aufgaben am Schluss dieses Buchs überprüfen Sie den Lernerfolg.

Die Technologie Java bildet den Schwerpunkt des dritten Teils »Plattform Java«. Dieser Teil stellt Ihnen zusätzlich vor, welche Gesetzmäßigkeiten Sie beim Programmieren beachten müssen, was Klassenbibliotheken sind und welche Vorteile sie haben. Zusätzlich erfahren Sie, wie Sie Programme testen, was Algorithmen sind und wie Sie sie programmieren.

Ein größeres Java-Projekt steht im Mittelpunkt des vierten Teils. Hier lernen Sie alle Elemente der vorigen Teile an einer Anwendung mit grafischer Oberfläche kennen. Das Projekt zeigt, wie man mit der Eclipse-Entwicklungsumgebung Stück für Stück eine größere Anwendung entwickelt.

Der fünfte Teil »Anhang« beschließt dieses Buch mit Lösungen zu den Aufgaben, mit Grundlagen der Informationsverarbeitung und einem Kapitel zu den häufigsten Fehlern, die bei der Arbeit mit Eclipse entstehen können.

## Rahmenhandlung

Als Rahmenhandlung habe ich dem Buch den (fiktiven) Programmierkurs »Java mit Eclipse« von Professor Roth mit vier Studentinnen und Studenten zugrunde gelegt. Den Programmierkurs begleitet der Roboter namens Robert und – neben vielen anderen – vor allem diese fünf Figuren durch das gesamte Buch:



Der Programmierkurs mit Lukas, Anna, Professor Roth, Julia und Florian

## Für wen ist das Buch?

Dieses Buch richtet sich an aktive Leser. Sie wollten nicht fertige Lösungen, sondern möchten selbst programmieren. Ohne dass Sie selbst aktiv programmieren und am Ball bleiben, bis Ihr selbstgeschriebenes Programm läuft, lernen Sie kein Java. Das Buch enthält verlockend viele fertig programmierte Beispiele, die Sie auf Knopfdruck ausführen können. Greifen Sie nur zu den Musterlösungen, wenn Sie nicht weiterkommen. Versuchen Sie zuerst, die Programme selbst einzugeben und aus den Fehlern zu lernen. Nur durch aktives Programmieren beherrschen Sie Java und die Eclipse-Entwicklungs Umgebung.

## Bonusmaterial

Das Buch enthält eine Fülle von Beispielen, die einfach in die Eclipse-Umgebung als Lösungen importiert werden können. Sie können Sie einfach von <https://plus.hanser-fachbuch.de> herunterladen (siehe auch Abschnitt 4.2.4, dort finden Sie im Abschnitt »Download der Beispielprogramme« nähere Informationen). Unter diesen Downloads finden Sie zudem ein Bonuskapitel, das aus Platzgründen nicht abgedruckt ist. Es erklärt die Programmierung von sogenannten Terminalprogrammen.

## Schriftkonventionen

Verschiedene Textteile sind zur besseren Lesbarkeit wie folgt hervorgehoben:

Textteil	Bedeutung
Datentypen im Fließtext	<i>Person</i>
Datentypen in Überschriften	»Person«
Schlüsselwörter im Fließtext	<i>implements</i>
Schlüsselwörter in Überschriften	»implements«
Variablen im Fließtext	<i>roland</i>
Variablen in Überschriften	»roland«
Fenster (grafische Oberfläche)	ECLIPSE IDE LAUNCHER
GUI-Element (grafische Oberfläche)	FINISH
Menü (grafische Oberfläche)	FILE
Menübefehl (grafische Oberfläche)	MENÜ → FILE → NEW → JAVA PROJECT
Dateien	<i>Beispielprogramme.zip</i>
Verzeichnispfade	<i>C:/Programme/Eclipse</i>
Listing (Quellcode von Beispielprogrammen)	<pre> 1 package programmierkurs; 2 public class Roboter { 3 (... ) 4 }</pre>
Programmausgabe	Ergebnis = true
URL	<i>http://eclipse.org</i>
(...)	Aus Platzgründen fehlt ein Teil des Quellcodes



## Herzlichen Dank!

Hiermit möchte ich mich bei allen bedanken, die mich beim Schreiben dieses Buchs unterstützt haben: dem Carl Hanser Verlag und meiner Lektorin Brigitte Bauer-Schiewek für das Vertrauen in meine Arbeit und die große Geduld. Der Herstellung des Hanser Verlags möchte ich für die professionelle  $\LaTeX$ -Vorlage danken und Herrn Korell für seine stets kompetente Unterstützung bei Fragen zum Satz dieses Buchs mit der  $\LaTeX$ -Vorlage.

Meine Frau Christiane hat mich wie immer sehr bei diesem Projekt unterstützt. Herzlichen Dank für Deine Hilfe! Danken möchte ich zudem Herrn Dubau, der mein Buchmanuskript von Rechtschreibfehlern befreit hat. Herzlichen Dank auch an Valery Kachaev, von dem die Vorlagen für die Roboter-Cartoons stammen.

## Kontakt

Trotz größter Sorgfalt lässt sich nicht immer verhindern, dass der eine oder andere Fehler in einem Buch übersehen wird. Wenn Sie Fehler finden, Verbesserungsvorschläge oder Fragen haben, senden Sie mir einfach eine E-Mail an [java-eclipse@steppan.net](mailto:java-eclipse@steppan.net). Ich werde Ihre Fragen möglichst schnell beantworten und versuchen, Ihre Verbesserungsvorschläge in kommenden Auflagen zu berücksichtigen. Die jeweils aktuellsten Ergänzungen und weitere Informationen finden Sie unter <http://www.steppan.net>. Nun wünsche ich viel Spaß beim Lesen und Entwickeln Ihrer Java-Programme mit Eclipse!

*Bernhard Steppan*

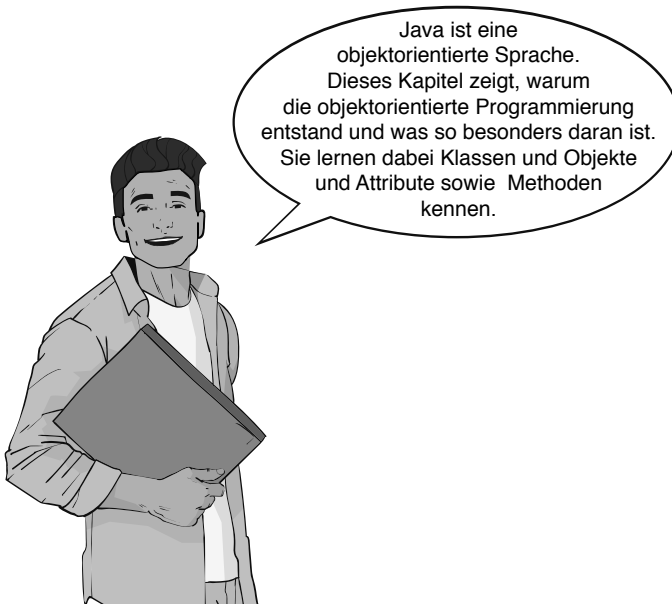
Wiesbaden im August 2020

# 3

## Objektorientierte Programmierung

### ■ 3.1 Einleitung

Java ist eine objektorientierte Programmiersprache. Daher ist es wichtig, genau zu verstehen, wie objektorientierte Programmierung funktioniert. In diesem Kapitel dreht sich alles um Objekte und Klassen, um Attribute und Methoden sowie darum, wie sich Objekte vor Übergriffen anderer feindlicher Objekte schützen lassen. Sie erfahren dabei auch, warum Objektorientierung entstand und was so besonders daran ist.



**Abbildung 3.1** Florian erklärt Ihnen, was das Besondere an objektorientierter Programmierung ist.

## ■ 3.2 Überblick

Wie kam das alles? Es begann alles Mitte der 60er-Jahre des 20. Jahrhunderts. Damals kam es zu einer Softwarekrise. Zu dieser Zeit stiegen die Anforderungen an Programme. Die Software wurde dadurch komplexer und fehlerhafter. Auf Kongressen diskutierten Experten die Ursachen der Krise und die Gründe für die gestiegene Fehlerrate.

Ein Teil der Softwareexperten kam zu dem Schluss, dass die Softwarekrise nicht mit den herkömmlichen Programmiersprachen zu bewältigen sei. Sie kritisierten an den herkömmlichen Programmiersprachen vor allem, dass sich die natürliche Welt bisher nur unzureichend abbilden lasse. Sie begannen deshalb, eine Generation von neuen Programmiersprachen zu entwickeln.

Alan Kay, der Erfinder der Programmiersprache »Smalltalk«, hat diese sechs Grundregeln für objektorientierte Sprachen aufgestellt.

```
Objektorientierte Programmierung {  
  1 Alles ist ein Objekt  
  2 Objekte haben ihren eigenen Speicher  
  3 Eine Klasse modelliert das gemeinsame  
    Verhalten ihrer Objekte  
  4 Jedes Objekt ist ein Exemplar seiner Klasse  
  5 Objekte kommunizieren über Nachrichtenaustausch  
  6 Ein Programm wird ausgeführt, indem dem ersten  
    Objekt die Steuerung übergeben und der Rest  
    als dessen Nachricht behandelt wird  
}
```



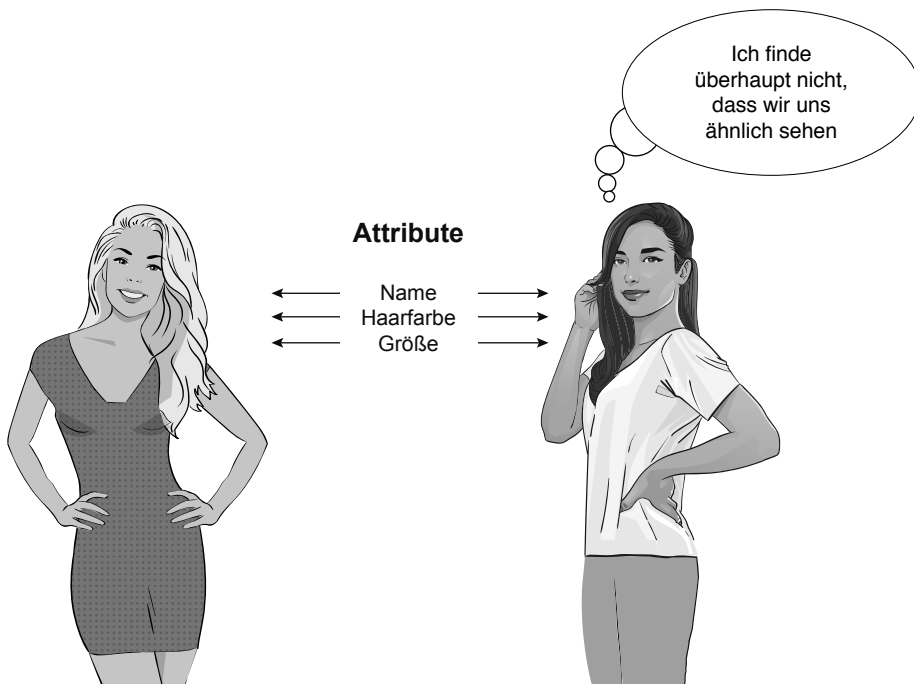
**Abbildung 3.2** Merkmale objektorientierter Sprachen

Die Experten begannen damit, natürliche Begriffe aus der Formenlehre der klassischen griechischen Philosophie für die neuen Programmiersprachen zu verwenden. Sie wandelten diese Bezeichnungen für die Programmierung ab (Abbildung 3.2). Da sich alles um den Begriff des Objekts dreht, nannten sie die neue Generation von Sprachen »objektorientiert«.

## ■ 3.3 Objekt

*Objekte* sind für ein Java-Programm das, was Zellen für einen Organismus sind: Aus diesen kleinsten Einheiten setzt sich ein Java-Programm zusammen. Wenn Sie eine Reihe von gleichartigen Objekten betrachten, fällt auf, dass ihr prinzipielles *Aussehen* gemeinsam ist. In der objektorientierten Programmierung bezeichnet man solche Objekte auch oftmals als *Exemplare* einer Klasse.

Als Beispiel soll wieder der Programmierkurs von Professor Roth dienen. An dem Programmierkurs nehmen Anna, Julia, Lukas und Florian teil. Greifen wir zunächst nur die Studentinnen Anna und Julia heraus. Beide Studentinnen sind Objekte mit vielen Gemeinsamkeiten. Gemeinsam ist ihnen zum Beispiel, dass sie Frauen sind, den gleichen Programmierkurs besuchen und an der gleichen Hochschule eingeschrieben sind.



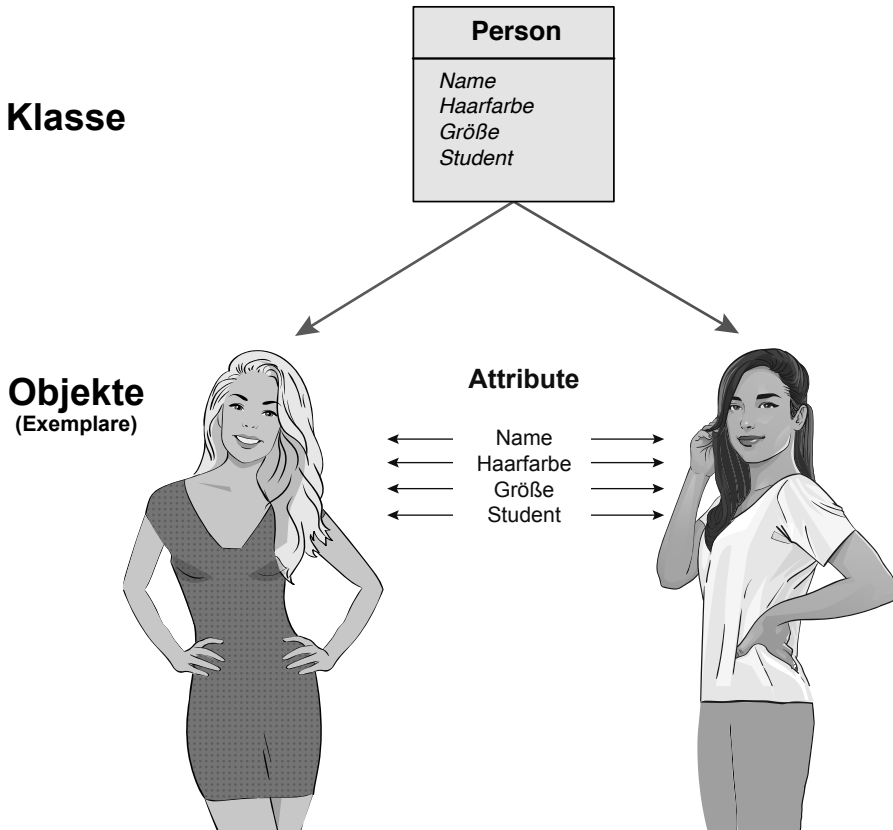
**Abbildung 3.3** Die »Objekte« Anna und Julia sind sich ähnlich, haben aber unterschiedliche Attribute.

Aus Sicht der objektorientierten Programmierung bedeutet das: Diese beiden Objekte sind *ähnlich*. Die Unterschiede zwischen diesen Objekten ergeben sich aus dem unterschiedlichen Wert ihrer *Attribute*. Beiden Studentinnen haben zum Beispiel einen verschiedenen Namen und unterschiedliche Haarfarbe sowie Größe (Abbildung 3.3).

Gleichartige Objekte haben also nur ihre *prinzipielle* Gestalt und bestimmte Fähigkeiten gemeinsam. Alles andere ist *individuell*. Die gemeinsame Gestalt und die gemeinsamen Fähigkeiten von Objekten legt der Bauplan der Objekte fest. Diesen Bauplan nennt die objektorientierte Programmierung *Klasse*.

## ■ 3.4 Klasse

Die *Klasse* ist es, die die prinzipielle Gestalt und die Fähigkeiten von Objekten wie den beiden Studentinnen prägt. Eine Klasse verhält sich zu einem Objekt wie der Bauplan eines Menschen zu einem realen Menschen. Die Klasse gibt verschiedenen Objekten der gleichen Art einen Obergriff. Man sagt auch, dass eine Klasse seine Objekte *klassifiziert* – daher auch der Name.



**Abbildung 3.4** Die Klasse »Person« liefert den Bauplan für die »Objekte« Anna und Julia.

### 3.4.1 Attribute

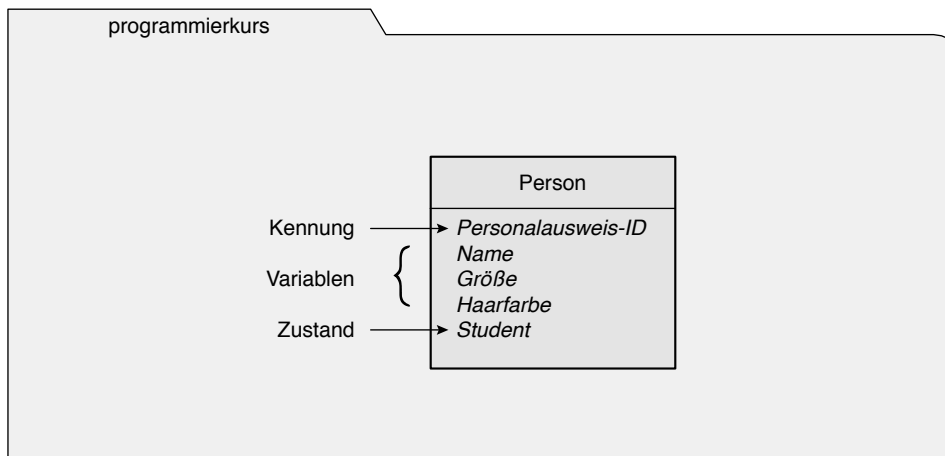
Unsere neue Klasse *Person* soll neben den Attributen *Name*, *Haarfarbe* und *Größe* noch zusätzlich das Attribut *Student* bekommen. Dieses Attribut legt fest, ob jemand Student ist oder nicht. Wenn aus dieser Klasse neue Personenobjekte entstehen, besitzen alle Exemplare einen individuellen *Namen*, eine individuelle *Haarfarbe*, eine individuelle *Größe* und einen Wert für die Markierung *Student* (Abbildung 3.4).

## Konstanten

Beispielsweise soll *Anna* über folgende Attribute verfügen: Name = Anna, Haarfarbe = blond, Größe = 1,71 m. Ihre Freundin aus dem Programmierkurs heißt Julia, hat die Haarfarbe braun und ist 1,72 m groß. Obwohl beide Personen nach dem gleichen Bauplan (Klasse) erzeugt worden sind, sind zwei deutlich unterschiedliche Objekte entstanden: Beide haben unterschiedliche Namen, haben unterschiedliche Haarfarbe und sind beide unterschiedlich groß.

## Zustände

Es ist Ihnen vielleicht aufgefallen, dass bei den bisherigen Attributen der beiden Personen einige mit festen Werten belegt waren, andere hingegen mit veränderlichen Werten. Die flexiblen Attribute beschreiben den *Zustand* des Objekts. Zum Beispiel beschreibt das Attribut *Student*, ob eine Person gerade an einer Hochschule eingeschrieben ist. Der Zustand eines Objekts kann sich im Laufe der Zeit ändern.



**Abbildung 3.5** Variablen, Zustände und Kennungen sind Attribute einer Klasse.

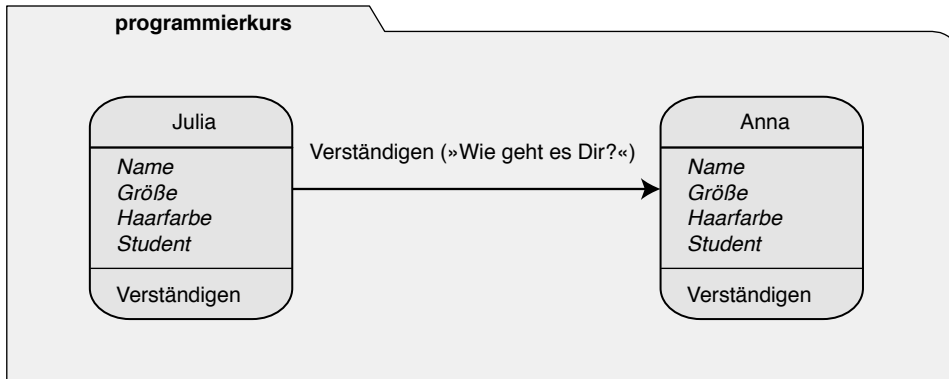
## Kennungen

Was würde passieren, wenn man die Objekte *Anna* und *Julia* so erzeugen würde, dass sie die gleiche *Größe*, die gleiche *Haarfarbe* und den gleichen Zustand *Student* besitzen? Wie könnte man sie dann unterscheiden? In diesem Fall haben beide Objekte zwar individuelle Werte für ihre Attribute bekommen, aber diese sind zufällig gleich. Damit gleichen sich auch die Objekte in einem Programm wie eineiige Zwillinge.

Um Objekte also besser zu unterscheiden, benötigt man so etwas wie einen genetischen Fingerabdruck. In der Programmierung vergibt der Entwickler eine sogenannte Kennung. Diese Kennung ist ein zusätzliches Attribut, bei dem darauf geachtet wird, dass es *eindeutig* ist. Erst die Kennung eines Objekts sorgt dafür, dass das Programm unterschiedliche Exemplare auch dann unterscheiden kann, wenn ihre Attribute zufällig die gleichen Werte besitzen.

### 3.4.2 Methoden

Angenommen, Sie wollen *Anna* mitteilen, dass sie auf eine Frage antworten soll. Im wirklichen Leben stellen Sie *Anna* einfach eine Frage oder schicken ihr eine Nachricht. In der objektorientierten Programmierung rufen Sie stattdessen eine Methode des Objekts *Anna* auf (Abbildung 3.6). Eine Methode ist der objektorientierte Begriff für eine Funktion. Der Begriff wurde eingeführt, um auszudrücken, dass eine objektorientierte Funktion viel leistungsfähiger ist als eine Funktion einer klassischen Programmiersprache.



**Abbildung 3.6** Objekte verständigen sich durch den Aufruf von Methoden.

Aber egal, wie der Begriff genannt wird, eines ist gleich: Verhaltensweisen wie *Verständigen* bestimmen die Fähigkeit eines Objekts, zu kommunizieren und Aufgaben zu erledigen. Objekte lassen sich also über Methoden steuern. Es existiert nicht nur eine Art von Methoden, sondern es gibt folgende fünf Grundtypen: *Konstruktoren* (»Erbauer«), *Destruktoren* (»Zerstörer«), *Änderungsmethoden* (»Setter«), *Abfragemethoden* (»Getter«) und *Operationen* (»Funktionen«).

#### Konstruktoren

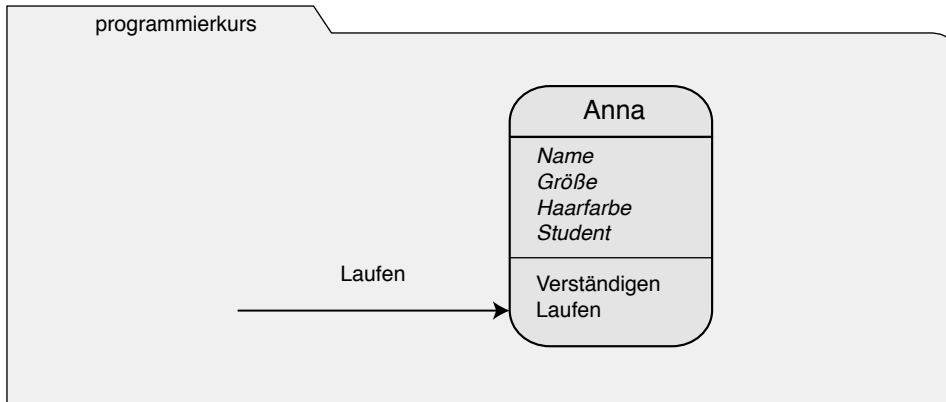
Wie im natürlichen Leben haben auch Objekte eines Computerprogramms einen Lebenszyklus. Sie werden geboren, also erzeugt, und irgendwann sterben sie, das heißt, sie werden gelöscht und ihr Speicher wird freigegeben. Die wichtigste Methode ist die, die ein Objekt erzeugt. Sie werden demzufolge auch »Konstruktoren« genannt. Sie konstruieren, das heißt erschaffen ein Objekt.

#### Destruktoren

Methoden, die ein Objekt zerstören, nennen sich in der objektorientierten Programmierung »Destruktoren«. In manchen Programmiersprachen können Sie diese Destruktoren direkt aufrufen und damit unmittelbar ein Objekt zerstören. In Java funktioniert dies nicht. Hier wird ein Objekt automatisch zerstört, wenn es nicht mehr benötigt wird. Dazu später mehr.

## Änderungsmethode

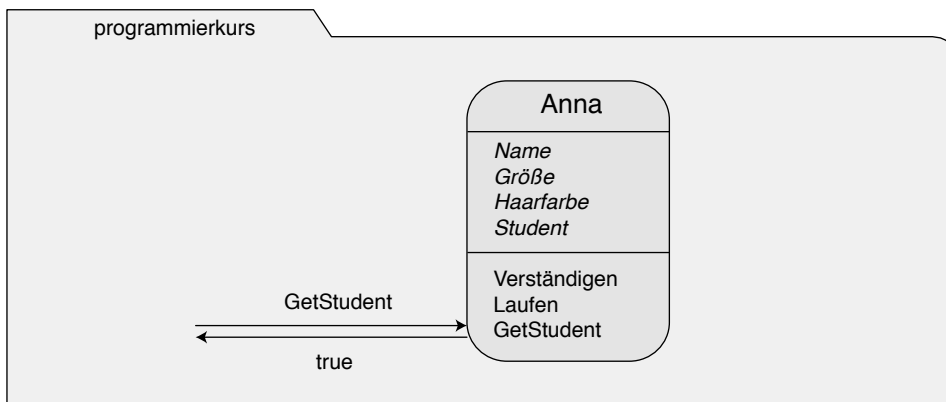
Methoden, die den Wert eines Attributs verändern, nennen sich »Änderungsmethoden« oder englisch »Setter«. Sie verändern den Zustand des Objekts. Mit einer solchen Methode lässt sich die Geschwindigkeit ändern, mit der *Anna* sich fortbewegt (Abbildung 3.7). Die entsprechende Methode nennt sich *Laufen* und verfügt über einen sogenannten Parameter, der den neuen Zustand, die *Geschwindigkeit*, vorgibt.



**Abbildung 3.7** Die Änderungsmethode »Laufen« ändert den Zustand von »Anna«.

## Abfragemethode

Abfragemethoden oder englisch »Getter« sind Methoden, die nur ein bestimmtes Attribut abfragen. Sie ändern nichts am Zustand des Objekts. Eine solche Methode wäre zum Beispiel die Abfrage, ob Anna an einer Hochschule eingeschrieben ist (Abbildung 3.8). Diese Methode besitzt einen sogenannten Rückgabewert: den Status des Studenten, der mit *true* oder *false* beantwortet werden kann.

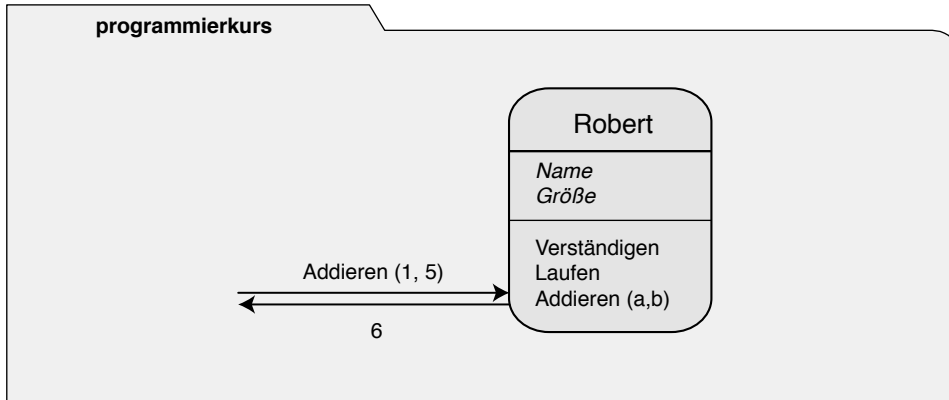


**Abbildung 3.8** Die Abfragemethode erfragt den Studentenstatus von Anna.



## Operationen

Methoden, die zum Beispiel nur eine Rechenoperation durchführen, werden in der objektorientierten Programmierung meistens als Operationen oder Funktionen bezeichnet. Sie dürfen trotzdem nicht mit den gleichnamigen Funktionen klassischer Programmiersprachen verwechselt werden, denn sie werden wie andere Methoden auch von Klasse zu Klasse weitervererbt (Abschnitt 3.6, »Vererbung«). Zudem lassen sie sich überladen und überschreiben (Abschnitt 3.13, »Polymorphie«).



**Abbildung 3.9** Die Operation »Addieren« liefert die Summe als Ergebnis zurück.

## ■ 3.5 Abstraktion

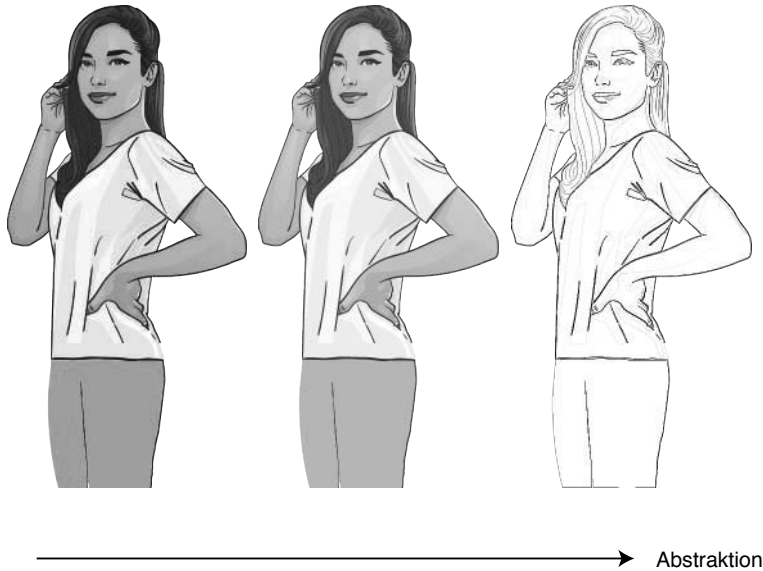
Vielleicht werden Sie jetzt sagen: »Das ist doch alles Unsinn. Die Fähigkeiten und Attribute einer Person sind viel komplexer und können nicht auf Größe und Farbe sowie auf Verständigen reduziert werden.« Das ist in der natürlichen Welt richtig, aber in der künstlichen Welt der Softwareentwicklung in der Regel völlig falsch.

Richtig wäre es nur dann, wenn man die Natur in einem Programm vollständig abbilden müsste. Aber für so eine übertriebene Genauigkeit gibt es bei der Programmierung selten einen Grund. Die objektorientierte Programmierung erleichtert eine möglichst natürliche Abbildung der realen Welt und fördert damit gutes Softwaredesign.

Sie verführt damit auch zu übertriebenen Konstruktionen. Die Kunst besteht darin, dem entgegenzusteuern und die Wirklichkeit so genau wie nötig, aber so einfach wie möglich abzubilden. Wie Sie später bei größeren Beispielprogrammen sehen werden, bereitet gerade die Analyse der für das Programm wesentlichen und richtigen Bestandteile oftmals große Probleme.

Wenn man innerhalb eines Programms nur die für die Funktionalität wesentlichen Teile programmiert, dann hat das praktische Gründe: Das Programm lässt sich schneller entwickeln, es wird billiger und schlanker. Somit benötigt es weniger Speicherplatz, und es wird in der Regel schneller ablaufen als ein Programm, das mit unnötigen Informationen überfrachtet ist.

Um diese Kompaktheit zu erreichen, ist es notwendig, die meist extrem komplizierten natürlichen Objekte und deren Beziehungen so weit es geht zu abstrahieren, also zu vereinfachen. Der Fachbegriff für diese Technik nennt sich demzufolge auch *Abstraktion* (Abbildung 3.10).



**Abbildung 3.10** Durch Abstraktion erhält man das Wesentliche einer Klasse.

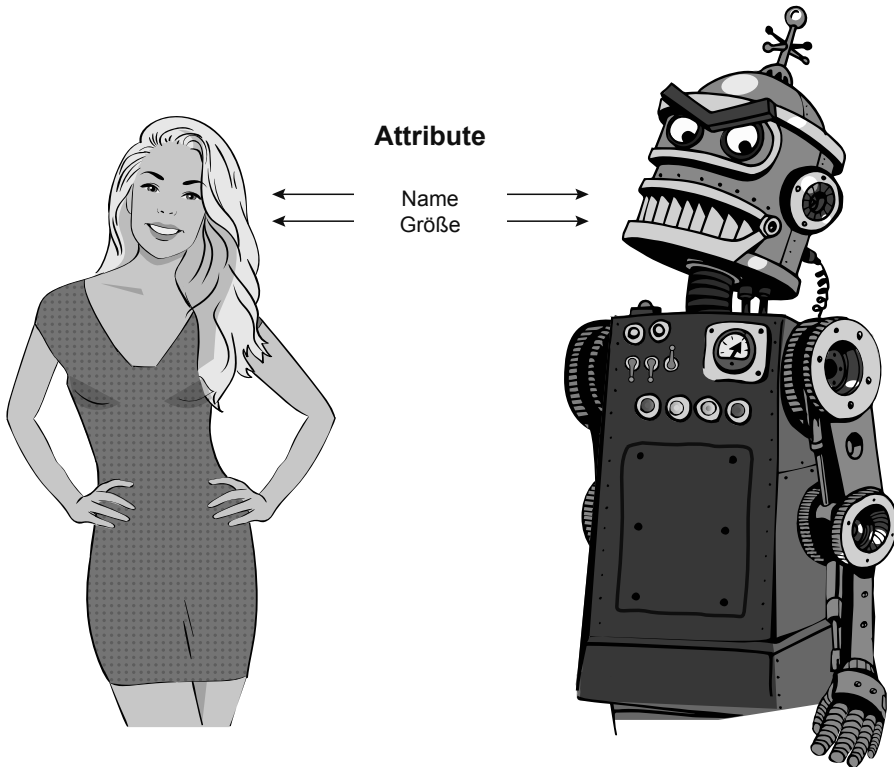
## ■ 3.6 Vererbung

Nach der Einführung von Klassen, Objekten, Methoden und Attributen wird es Zeit, diese neuen Begriffe in Zusammenhang mit dem Begriff der *Vererbung* zu bringen. Vererbung gestattet es, das Verhalten zwischen Klassen und damit auch zwischen Objekten mithilfe eines Bauplans zu übertragen. Die Vererbung kopiert Attribute und Methoden der Basisklasse auf die abgeleitete Klasse.

Um das zu verdeutlichen, wieder ein Beispiel: Menschen und Roboter sind sich in mancher Hinsicht ähnlich, aber in vielfältiger Hinsicht doch extrem verschieden. Diese Unterschiede sind von anderer Güte als die Unterschiede zwischen zwei Menschen: Menschen und Roboter haben eine deutlich unterschiedliche Gestalt (Abbildung 3.11).

Dass Menschen im Sinne der Formenlehre eine andere Gestalt besitzen als Roboter, muss man angesichts der wuchtigen Erscheinung von Robert eigentlich nicht besonders betonen. Um die Unterschiede auf den Punkt zu bringen, hilft es, wenn Sie einfach einmal versuchen, die auf den vorhergehenden Seiten aufgestellten Attribute von Personen mit denen eines Roboters in Einklang zu bringen. Wie Sie sehen werden, funktioniert das nur für einen bestimmten Teil der Attribute.

Was bedeutet das für die Programmierung? Das bedeutet, dass Sie auf die gerade gezeigte Weise herausfinden können, ob Objekte zu einer gemeinsamen Klasse gehören. In allen



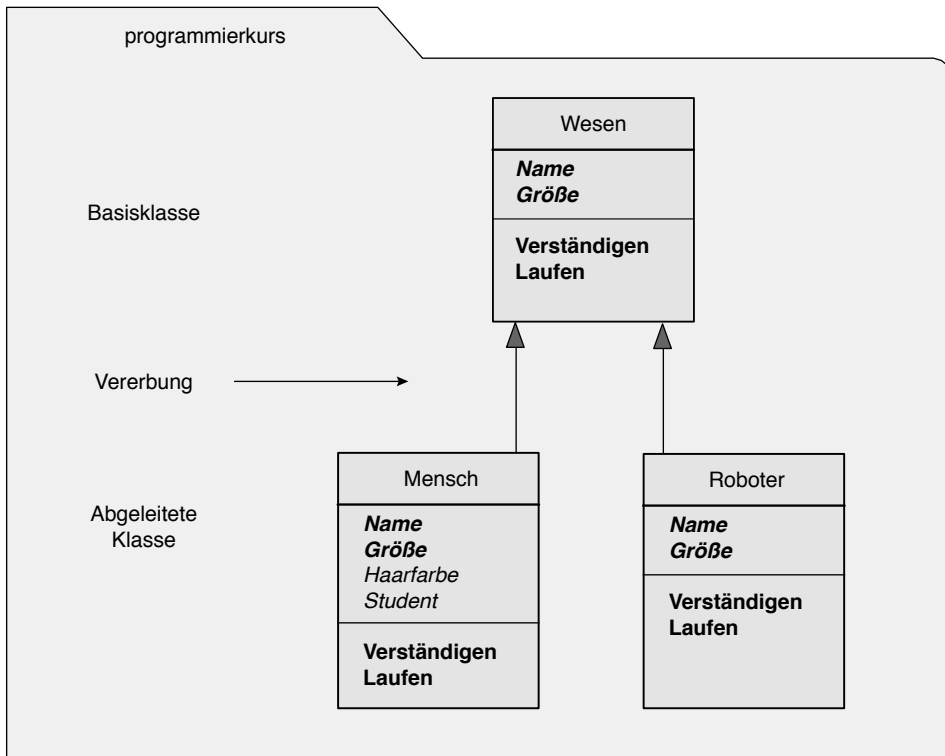
**Abbildung 3.11** Objekte verschiedener Klassen unterscheiden sich in ihrer Form.

Fällen, in denen es auf die Gemeinsamkeiten ankommt, sollte man den Objekten eine gemeinsame Klasse zuordnen. In allen Fällen, in denen es auf die Unterschiede zwischen Objekten ankommt, weist man ihnen besser getrennte Klassen zu.

### 3.6.1 Basisklassen

Bis jetzt haben wir die Unterschiede zwischen Mensch und Roboter betont. Was passiert, wenn wir es einmal von der anderen Seite betrachten: Was haben ein Mensch wie Anna und eine humanoide Maschine wie Robert gemeinsam? Aus Sicht der Programmierung hat jeder Mensch wie Anna und jeder Roboter wie Robert eine bestimmte Größe. Sie können beide laufen und sich verständigen.

Die genannten Eigenschaften teilen sie mit einer Vielzahl von Lebewesen. Sind Roboter Lebewesen? Wohl eher nicht. Die meisten Menschen würden ihnen sicher die Fähigkeit zu leben absprechen. Man könnte sie aber als künstliche Wesen oder Maschinenwesen bezeichnen. Aus Sicht der objektorientierten Programmierung bedeutet das, dass wir Mensch und Roboter einer gemeinsamen Basisklasse *Wesen* zuordnen könnten. Abbildung 3.12 zeigt, wie eine solche Basisklasse aussehen würde.

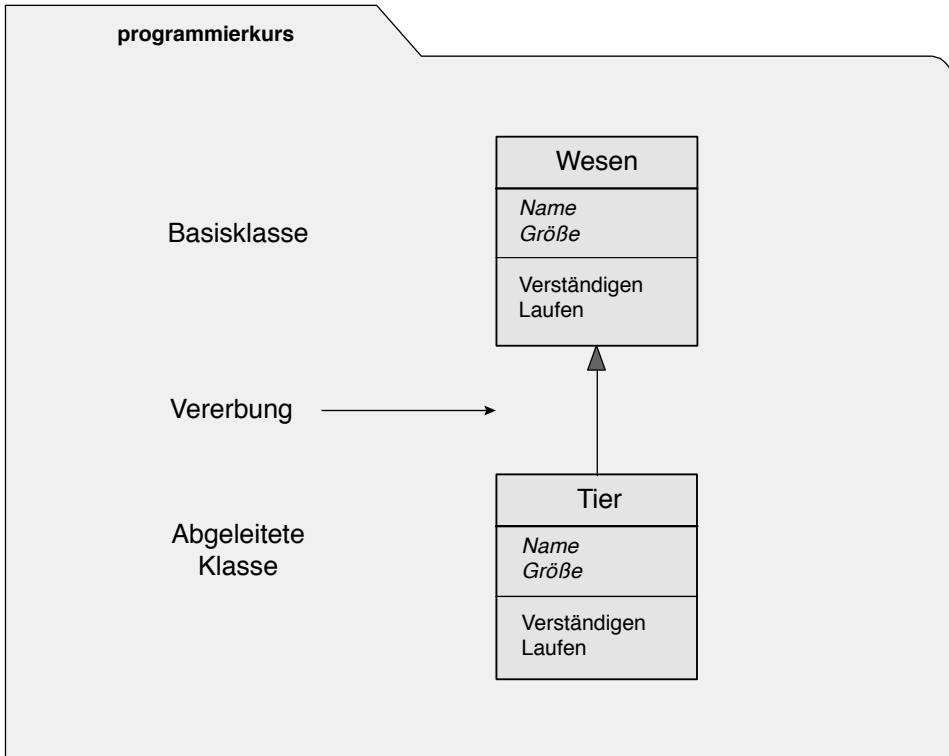


**Abbildung 3.12** Die Basisklasse überträgt Basiseigenschaften und -verhalten.

Beachten Sie besonders die fett hervorgehobenen Wörter: Damit man nicht für jede der Klassen *Mensch* und *Roboter* die Attribute *Name* und *Größe* sowie das Verhalten *Verständigen* und *Laufen* neu programmieren muss, bietet es sich an, dieses Verhalten in eine Basisklasse zu verlagern. Die fett gedruckten Attribute und Methoden sind aus Sicht der objektorientierten Programmierung die *Gemeinsamkeiten* beider Klassen. Die gemeinsamen Attribute der Basisklasse erleichtern nicht nur die Programmierung. Sie vereinheitlichen auch alle abgeleiteten Klassen.

### 3.6.2 Abgeleitete Klassen

Angenommen, Sie möchten eine neue Klasse namens *Tier* auf Basis der Klasse *Wesen* erzeugen. In der objektorientierten Programmierung würde man sagen, man leitet von der Klasse *Wesen* eine neue Klasse namens *Tier* ab. Wie in der Ahnenforschung bei Stammbäumen sagt man auch, die Klasse *Tier* stammt von der Klasse *Wesen* ab. Die neue Klasse *Tier* würde, wie schon zuvor die Klassen *Mensch* und *Roboter*, die Attribute *Name*, *Größe* sowie das Verhalten mit *Verständigen* und *Laufen* von der Basisklasse *Wesen* übernehmen. Attribute und Verhalten vererben sich also (Abbildung 3.13).



**Abbildung 3.13** Die neue Klasse »Tier« ist eine von »Wesen« abgeleitete Klasse.

### 3.6.3 Mehrfachvererbung

In der Natur ist sie üblich, in der Programmiersprachen Java jedoch aus gutem Grund nicht erlaubt: die Mehrfachvererbung. Sie wäre dann praktisch, wenn Sie zwei Klassen verschmelzen wollten, zum Beispiel die Klasse *Mensch* mit der Klasse *Pferd*. Die neue Kreuzung *Kentaur* würde Attribute und Verhalten beider Basisklassen erben (Abbildung 3.14). Aber welche Attribute und welches Verhalten? Sollen sich Kentauren verständigen und laufen wie Menschen oder wie Pferde? Können Kentauren Studenten sein?

So schön das Beispiel in der Mythologie ist, bei derartigen Szenarien kommt die Softwareentwicklung an die Grenze des technisch Sinnvollen. Es ist eben nicht sinnvoll, Erbinformationen nach dem Zufallsprinzip zu übertragen, um die Natur zu imitieren. Der Anwender wünscht sich im Regelfall Programme, die über definierte Eigenschaften verfügen und deren Verhalten vorhersehbar ist. Aus den genannten Gründen haben sich die Entwickler der Programmiersprache Java bewusst gegen die konventionelle Mehrfachvererbung entschieden.

Wie Sie trotzdem mehrere Basisklassen ohne Nebenwirkungen miteinander verbinden können, stellt Ihnen Kapitel 9, »Klassen und Objekte«, Abschnitt 9.5, »Interfaces«, vor.

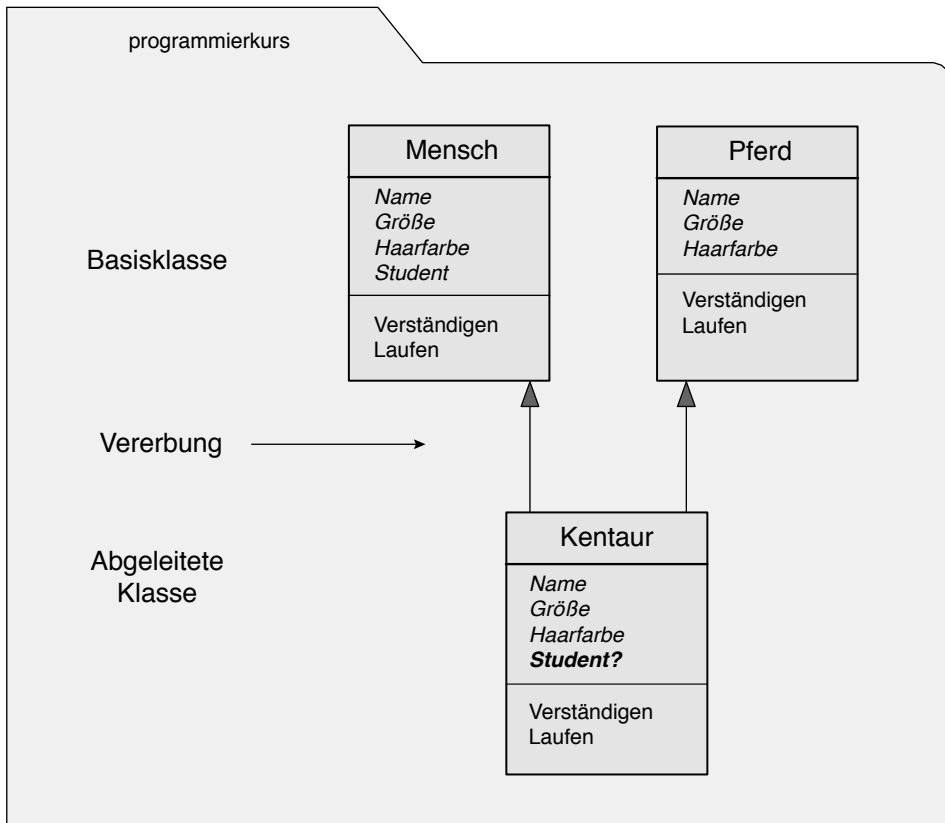


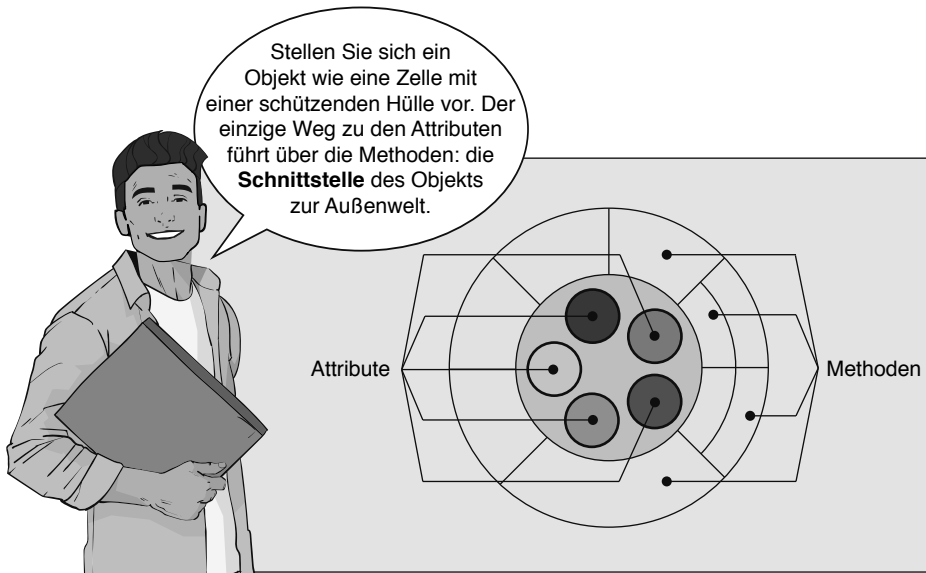
Abbildung 3.14 Mehrfachvererbung am Beispiel einer Kreuzung

## ■ 3.7 Sichtbarkeit

Eines der wichtigsten Merkmale objektorientierter Sprachen ist der Schutz von Objekten und ihren Attributen vor unerwünschtem Zugriff. Vielleicht erinnern Sie sich noch an den Anfang dieses Kapitels. Die objektorientierte Programmierung wurde erfunden, um die Softwarekrise zu überwinden, die durch fehlerhafte Software ausgelöst wurde. Die Software sollte durch die neue Programmierung robuster werden.

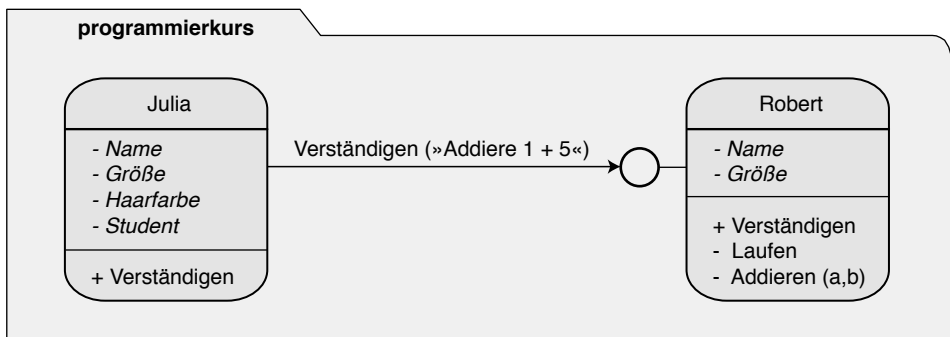
Aus dem Grund besitzt jedes Objekt eine Art von Kapsel, die die Daten und Methoden des Objekts schützt. Die Kapsel versteckt die Teile des Objekts, die von außen nicht oder nur durch bestimmte andere Objekte erreichbar sein sollen. Die Stellen, an denen die Kapsel durchlässig ist, nennen sich *Schnittstellen*. Die Idee des Ganzen ist, nur die Informationen zu einem Objekt durchzulassen, die es unbedingt erreichen müssen. Daher lässt sich jedes Objekt wie eine Zelle gestalten (Abbildung 3.15).

Die wichtigste Schnittstelle einer Klasse ist sein Konstruktor. Nehmen wir als Beispiel die Klasse *Roboter*. Über den Konstruktor dieser Klasse lässt sich das Objekt *Robert* erzeugen. Ein anderes Beispiel für eine solche Schnittstelle ist die Methode *Verständigen* der Klasse



**Abbildung 3.15** Wie eine Zelle schützt das Objekt seine Attribute vor unerwünschten Zugriffen.

*Roboter*. Sie ist öffentlich zugänglich. Im Gegensatz dazu ist seine interne Methode *Addieren* nicht öffentlich zugänglich. Anna kommuniziert mit *Robert* über diese Schnittstelle und teilt darüber *Robert* mit, was er berechnen soll (Abbildung 3.16).



**Abbildung 3.16** Objekte kommunizieren nur über ihre Schnittstellen.

Sämtliche Teile, auf die man von außen zugreifen darf, sind mit einem Pluszeichen markiert. Alle Teile, die von außen nicht manipuliert werden können, sind mit einem Minuszeichen versehen. Das Objekt *Anna* darf aber hierbei nicht sämtliche Daten von *Robert* über diese Schnittstelle verändern. Zum Beispiel soll es Anna keinesfalls erlaubt sein, den Namen des Roboters zu ändern. Gäbe es eine öffentlich zugängliche Methode wie zum Beispiel *Umbenennen*, so könnte sie *Robert* damit verändern. So ist das aber nicht gestattet.

## ■ 3.8 Beziehungen

Klassen und deren Objekte unterhalten in einem Programm die unterschiedlichsten Beziehungen untereinander. In den vorangegangenen Abschnitten haben Sie bereits mehrere Formen der Beziehungen kennengelernt. Grundsätzlich gibt es Beziehungen ohne Vererbung und Beziehungen mit Vererbung.

### 3.8.1 Beziehungen ohne Vererbung

Die objektorientierte Programmierung nimmt es mit Beziehungen sehr genau. Sie kennt gleich drei verschiedene Arten von Beziehungen ohne Vererbung (Abbildung 3.17).



**Abbildung 3.17** Abseits der Vererbungsbeziehung gibt es drei Beziehungstypen.

#### 3.8.1.1 Assoziation

*Assoziation* ist die einfachste Form einer Beziehung zwischen Klassen und Objekten. Die Abhängigkeiten sind bei dieser Beziehungsart im Vergleich zur Vererbung gering. Man sagt auch, die Objekte sind lose gekoppelt.

Eine Assoziation besteht zum Beispiel, wenn ein Objekt namens *Anna* einem Objekt namens *Robert* die Botschaft *Addieren* sendet (Abbildung 3.18). Die beiden Objekte *Anna* und *Robert* existieren getrennt und erben nichts voneinander.

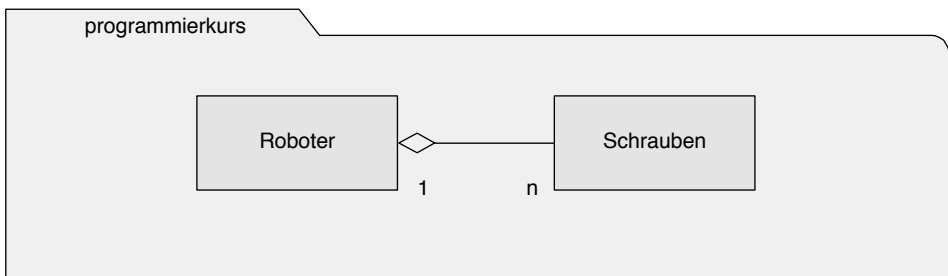




**Abbildung 3.18** Eine einfache Assoziation zwischen Mensch und Roboter

### 3.8.1.2 Aggregation

Eine Steigerung der Assoziation ist die Aggregation. Eine solche Beziehung besteht dann, wenn sich ein Objekt aus anderen Objekten zusammensetzt. Zum Beispiel soll ein Roboter aus einer nicht näher bestimmten Anzahl von Schrauben bestehen (Abbildung 3.19). Das bedeutet zum Beispiel, dass ein Roboter eine »Besteht-aus-Beziehung« zur Schraube unterhält.



**Abbildung 3.19** Aggregation zwischen Roboter und Schraube

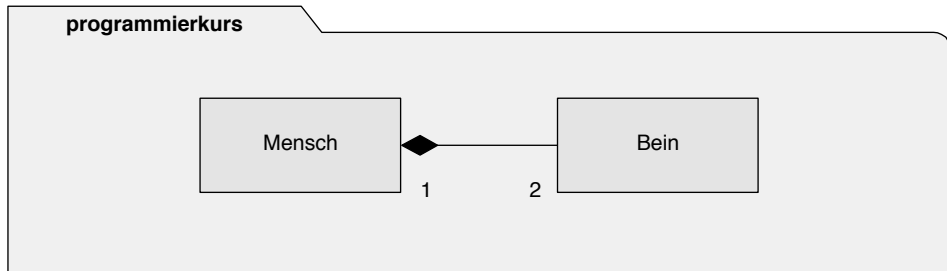
Diese Beziehung ist aber von einer völlig anderen Qualität als im vorangegangenen Beispiel zwischen einem Menschen und einem Roboter. Während Mensch und Roboter allein und unabhängig voneinander existieren können, setzt sich der Roboter (unter anderem) aus Schrauben zusammen. Wichtig ist hierbei wieder, dass beide Objekte nichts voneinander erben und jedes Schrauben-Objekt auch allein lebensfähig ist. Dieses Beispiel unterscheidet sich von der strengeren Komposition.

### 3.8.1.3 Komposition

Die stärkste Form der Beziehungen, die nicht auf Vererbung beruhen, stellt die *Komposition* dar. Wie bei der Aggregation liegt wieder eine »Besteht-aus-Beziehung« vor, sie ist aber im Gegensatz zur Aggregation abermals verschärft. Die Abhängigkeiten sind nochmals stärker.

Ein Beispiel für eine Komposition ist das Verhältnis zwischen einem Mensch und seinen zwei Beinen. Hier besteht glücklicherweise eine sehr enge Beziehung, denn ein Bein ist – im Gegensatz zur Schraube – als selbstständiges Objekt vollkommen sinnlos. Bei der

Erzeugung eines Menschen-Objekts bekommt dieses automatisch zwei individuelle Beine, die im Zusammenhang mit anderen Objekten nicht verwendet werden können.



**Abbildung 3.20** Ein Mensch und seine zwei Beine als Komposition

Menschenbeine sind also ohne ein geeignetes Objekt der Klasse *Mensch* nicht lebensfähig. Wenn ein Menschen-Objekt stirbt, so sterben auch seine Menschenbeine.

## 3.8.2 Vererbungsbeziehungen

Vererbungsbeziehungen nennen sich auch Generalisierung (Verallgemeinerung) oder Spezialisierung (Verfeinerung). Dies sind nicht etwa Unterarten der Vererbung, sondern alternative Begriffe für Vererbungsbeziehungen. Welchen der zwei alternativen Begriffe man verwenden möchte, hängt vom Blickwinkel ab, aus dem man die Vererbungsbeziehung betrachtet.

### 3.8.2.1 Generalisierung

Wenn Sie die Basisklasse aus dem Blickwinkel der abgeleiteten Klasse betrachten wollen, ist Generalisierung der passende Begriff dazu. Zum Beispiel ist die Klasse *Wesen* eine Generalisierung der Klassen *Mensch* und *Roboter*. Mit anderen Worten: Die Klasse *Wesen* ist der allgemeine Begriff (= Generalisierung) für die Klassen *Mensch* und *Roboter*.

### 3.8.2.2 Spezialisierung

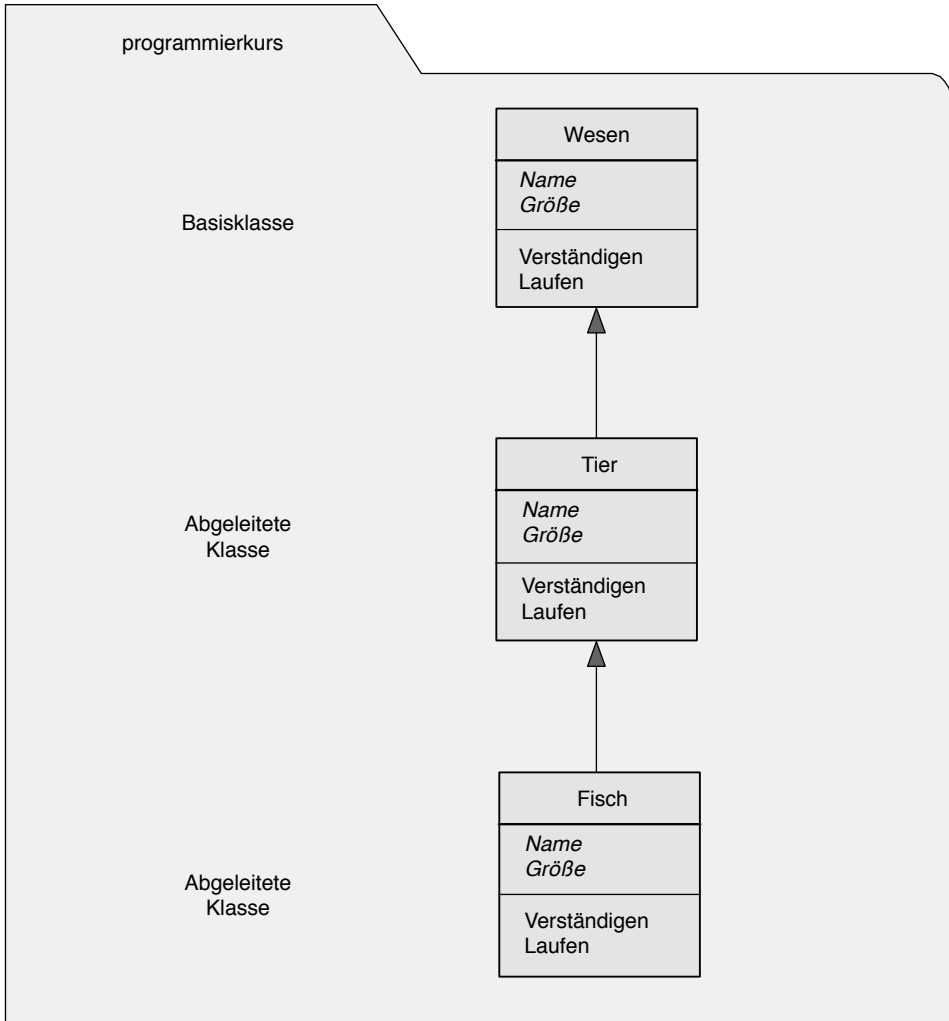
Wenn Sie die abgeleitete Klasse aus dem Blickwinkel der Basisklasse betrachten wollen, ist Spezialisierung der passende Begriff dazu. Zum Beispiel sind die Klassen *Mensch* und *Roboter* eine Spezialisierung der Klasse *Wesen*. Mit anderen Worten: Die Klassen *Mensch* und *Roboter* stellen eine Verfeinerung der Klasse *Wesen* dar.

### 3.8.2.3 Vererbung kann problematisch sein

Vererbungsbeziehungen stellen eine sehr starke Kopplung zwischen Klassen und damit auch zwischen Objekten her. Eine solch starke Kopplung hat nicht nur Vorteile, sondern auch gravierende Nachteile, wie das folgende Beispiel zeigt:

Eine Klasse namens *Fisch* soll aus der Klasse *Tier* erzeugt werden, die wiederum von *Wesen* abstammt (Abbildung 3.21). Die neue Klasse erbt die Attribute *Name* und *Größe* sowie die

Methoden *Verständigen* und *Laufen*. Moment mal: *Verständigen* und *Laufen*? Hier kommt man ins Grübeln. Können sich Fische verständigen? Vielleicht. Aber laufen, bis auf wenige Ausnahmen, können Fische sicher nicht.

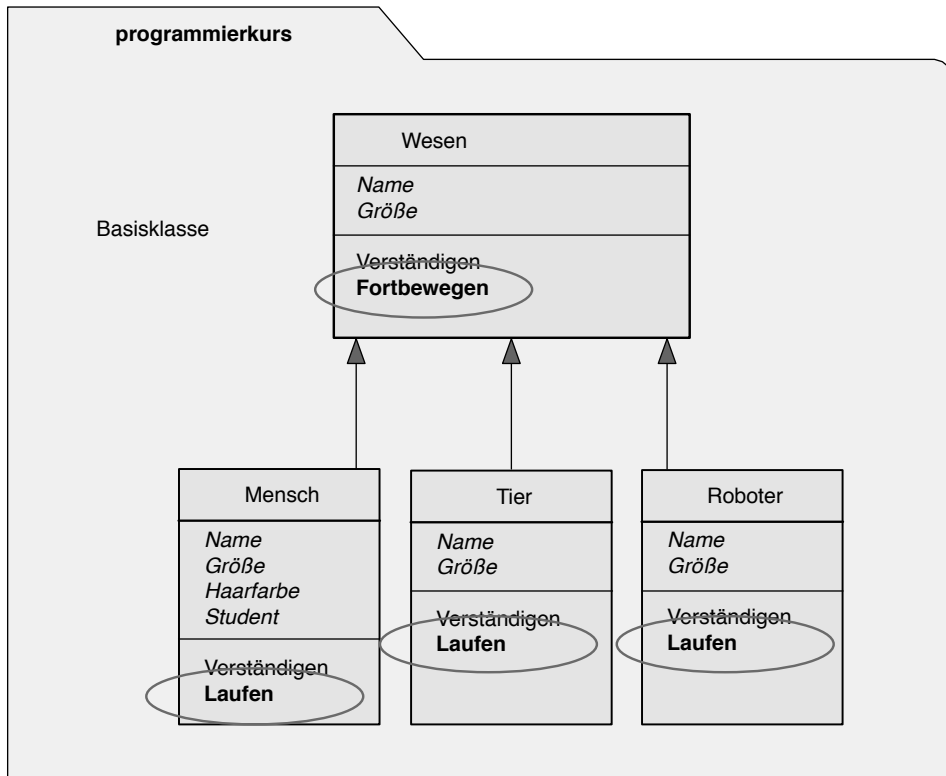


**Abbildung 3.21** Durch Vererbung vererben sich auch Designfehler.

Hier ist genau das passiert, was tagtäglich zu den Problemen der objektorientierten Programmierung gehört: Die Funktionalität der Basisklasse ist nicht ausreichend analysiert worden. Vereinfacht gesagt: Hier liegt ein Designfehler vor, den man dadurch beheben muss, dass man zumindest die Methode *Laufen* durch die Methode *Fortbewegen* ersetzt. Aber das hätte für die beiden Klassen *Mensch* und *Roboter* einige Konsequenzen.

## ■ 3.9 Designfehler

Sie können sich vielleicht vorstellen, dass es sehr unangenehm ist, wenn die Basisklasse aufgrund eines Designfehlers geändert werden muss. Durch die starke Kopplung zwischen Basisklasse und abgeleiteter Klasse pflanzen sich etwaige Änderungen lawinenartig in alle Programmteile fort, in denen Objekte des Typs *Mensch* und *Roboter* mit der Methode *Laufen* verwendet wurden. An allen Stellen des Programms, wo die Methode *Laufen* der Klasse *Wesen* verwendet wurde, muss sie durch die Methode *Fortbewegen* ersetzt werden (Abbildung 3.22)



**Abbildung 3.22** Durch Redesign lassen sich Fehler bei der Vererbung beheben.

Im Fall von Designfehlern stellt sich die Technik der Vererbung als großer Nachteil heraus. Vererbung hat neben diesem Manko auch den Nachteil, dass sich nicht nur Designfehler, sondern alle anderen vorzüglich gestalteten, aber unerwünschten Teile der Basisklasse in die abgeleiteten Klassen in Form von Ballast übertragen: Die Nachkommen solcher übergewichtiger Klassen werden immer fetter und fetter. Daher sollten Sie Vererbung stets kritisch betrachten, sparsam einsetzen und wirklich nur dort verwenden, wo sie sinnvoll ist.

## ■ 3.10 Umstrukturierung

Aber zurück zu den Designfehlern. Wie geht man mit Fehlern dieser Art um? Sie sind trotz der Vererbung heute kein so großes Problem mehr wie noch vor ein paar Jahren. Mit einem Werkzeug wie Eclipse ist es relativ leicht, die notwendige Umstrukturierung (Refactoring) vorzunehmen. Allerdings sollten Sie Software möglichst nur während der Analyse- und Designphase der Software umstrukturieren. Als Regel gilt: Je später Änderungen vorgenommen werden, desto höher ist der damit verbundene Aufwand. Kapitel 19, »Entwicklungsprozesse«, beleuchtet das Thema nochmals ausführlicher.

## ■ 3.11 Modellierung

Um solche Designfehler und damit kostspielige Umstrukturierungen zu vermeiden, ist es bei größeren Projekten sinnvoll, ein Modell der Software zu entwerfen. Genauso wie man im Automobilbau vor jedem neu zu konstruierenden Automobil ein Modell entwickelt, ist es auch in der Softwareentwicklung sinnvoll, ein Modell zu konstruieren, bevor man mit der eigentlichen Umsetzung des Projekts beginnt. Ein Modell, das eine getreue Nachbildung eines kompletten Ausschnitts der Software darstellt, nennt sich Prototyp (Muster, Vorläufer).

## ■ 3.12 Persistenz

Ein Programm erzeugt Objekte, die an ihrem Lebensende wieder zerstört werden. Diese Objekte bezeichnet man als transient, also flüchtig. Manchmal ist aber ein »Leben nach dem Tod« auch für Objekte erstrebenswert. Sie sollen auch dann wieder zum Leben erweckt werden, wenn das Programm beendet ist und der Anwender des Programms nach Hause geht. Am nächsten Tag startet der Anwender das Programm erneut und möchte mit dem gleichen Objekt weiterarbeiten.

Solche »unsterblichen« Objekte bezeichnet man als persistent (dauerhaft). Das bedeutet nichts anderes, als dass sie in geeigneter Form gespeichert werden. Sie befinden sich dann in einer Art Tiefschlaf in einer Datei auf einer Festplatte oder im Verbund mit anderen Objekten in einer Datenbank.

## ■ 3.13 Polymorphie

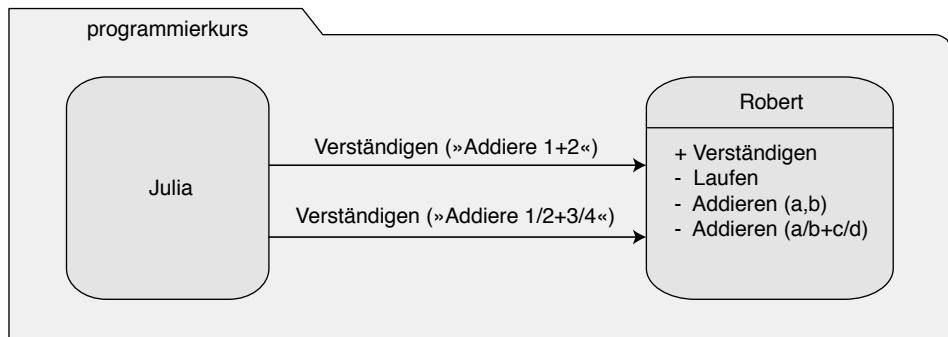
Der Name Polymorphie kommt aus dem Griechischen und bedeutet so viel wie Vielgestaltigkeit, Verschiedengestaltigkeit. Der Begriff klingt mehr nach Mineralienkunde als nach Informatik, und so wundert es Sie vielleicht auch nicht, dass der Chemiker Mitscherlich die Polymorphie bei Mineralien Anfang des 19. Jahrhunderts entdeckte. Er stellte fest, dass

manche Mineralien wie Kalziumcarbonat ( $\text{CaCO}_3$ ) unterschiedliche Kristallformen annehmen können, ohne ihre chemische Zusammensetzung zu ändern. Das bedeutet, sie können je nach Druck und Temperatur eine verschiedene Gestalt annehmen.

Alles sehr schön bis jetzt, aber was hat das mit objektorientierter Programmierung zu tun? Das bedeutet auf keinen Fall, dass ein Objekt wie *Robert* so radikal seine Form verändern kann wie ein Mineral. Es bedeutet, dass *Robert* bei geschickter »Programmierung« situationsbedingt verschieden reagieren kann. Klingt wie Zauberei, ist es aber nicht.

### 3.13.1 Statische Polymorphie

Stellen Sie sich vor, das Objekt *Anna* teilt dem Objekt *Robert* mit, dass *Robert* eine Addition mit zwei ganzzahligen Werten durchführen soll. Was wird passieren? – Natürlich ist es für *Robert* kein Problem, diese Werte zu addieren und *Anna* das Ergebnis zu nennen. Was würde aber passieren, wenn *Anna* abermals *Robert* mitteilt, er solle addieren, und zwar mit Bruchzahlen? Entweder würde *Robert* die Aufgabe zerlegen oder er würde eine Addition direkt mit Bruchzahlen durchführen, weil er eine interne Methode dafür hätte (Abbildung 3.23).



**Abbildung 3.23** »Robert« verfügt über zwei verschieden gestaltete Methoden namens »Addieren«.

Damit *Robert* den verschiedenen Anweisungen von *Anna* Folge leisten kann, benötigt er Methoden »unterschiedlicher Gestalt«. Er benötigt eine Methode, die auf zwei Parameter Geschwindigkeit reagiert, und eine Methode, die auf drei Parameter reagiert. Obwohl die Methoden den gleichen Namen tragen, führen sie zu einer unterschiedlichen Verarbeitung durch das Objekt *Robert*. Der Fachausdruck für diese Technik heißt *Überladen*.

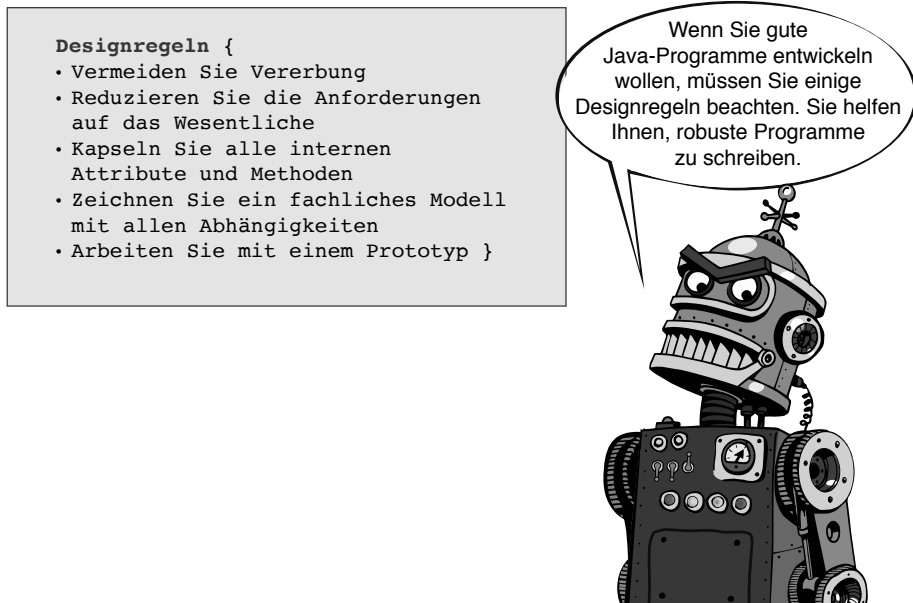
### 3.13.2 Dynamische Polymorphie

Anders als bei der Mehrfachvererbung sieht es aus, wenn man Eigenschaften der Basisklasse bei der Vererbung bewusst umgehen möchte. Dazu möchte ich nochmals auf das Beispiel der Basisklasse *Wesen* zurückgreifen. Angenommen, Sie möchten in der abgeleiteten Klasse *Roboter* bestimmen, auf welche Weise sich Roboter-Objekte verständigen. Dazu *überschreiben* Sie die Methode *Verständigen* und legen die Art und Weise des Verständigens in der Klasse *Roboter* für die abgeleiteten Objekte fest.

Das Überschreiben von Methoden ist ein sehr mächtiges Mittel der objektorientierten Programmierung. Es erlaubt Ihnen, unerwünschte Erbinformationen teilweise oder ganz zu unterdrücken und damit eventuelle Designfehler – in Grenzen – auszugleichen beziehungsweise Lücken in der Basisklasse zu füllen. Dabei ist die Technik extrem simpel. Es reicht aus, eine identische Methode in der abgeleiteten Klasse *Roboter* zu beschreiben, damit sich Objekte wie *Robert* »plötzlich« anders verhalten.

## ■ 3.14 Designregeln

Objektorientierte Programme sind keine Garantie für sauber strukturierte und logisch aufgebaute Programme. Die objektorientierte Programmierung erleichtert gutes Softwaredesign, sie erzwingt es jedoch nicht. Daher sollten Sie einige Grundregeln beachten. Sie erleichtern es Ihnen, schlanke, schnelle und robuste Java-Programme zu entwickeln (Abbildung 3.24).



**Abbildung 3.24** Robert empfiehlt einige Grundregeln für gutes Design.

## ■ 3.15 Zusammenfassung

Die objektorientierte Programmierung war eine Antwort auf die Softwarekrise in der Mitte der 60er-Jahre des letzten Jahrhunderts. Durch Objektorientierung lässt sich die natürliche Welt leichter in Computerprogrammen umsetzen. Diese objektorientierten Computerprogramme bestehen aus einem oder mehreren Objekten.

Die Objekte haben ihren eigenen Speicher. Klassen modellieren das gemeinsame Verhalten ihrer Objekte. Jedes Objekt ist hierbei ein Exemplar seiner Klasse. Objekte verständigen sich über Nachrichtenaustausch. Ein Programm wird ausgeführt, indem dem ersten Objekt die Kontrolle übergeben und der Rest als dessen Nachricht behandelt wird.



Objektorientierte Programmierung ist kein Allheilmittel. Sie unterstützt gutes Design, ohne es zu erzwingen. Es ist deshalb notwendig, auf sauberes Design zu achten, wenn man mit objektorientierter Programmierung erfolgreich sein will.

**Abbildung 3.25** Florian fasst die Besonderheiten der objektorientierten Programmierung zusammen.

Ein Objekt lässt sich mit einem natürlichen Lebewesen vergleichen und verfügt über eine Gestalt und Fähigkeiten. Die Gestalt prägen Attribute, während die Fähigkeiten von Methoden bestimmt sind. Beide Bestandteile eines Objekts sind in der Klasse festgelegt, von der ein Objekt abstammt. Sie liefert den Bauplan für gleichartige Objekte.



## ■ 3.16 Aufgaben

- Aufgabe 1: Konzipieren Sie eine Adressdatenbank der Studenten und Studentinnen sowie Mitarbeiter einer Hochschule. Die Datenbank soll dem Sekretariat helfen, Personen zu finden und sie anzuschreiben oder anzurufen. Orientieren Sie sich hierbei an Abbildung 3.26. Überlegen Sie sich passende Personenobjekte mit Attributen für Mitarbeiter sowie Studenten und zeichnen diese in ein Objektdiagramm.



**Abbildung 3.26** Verschiedene Personenobjekte an einer Hochschule

- Aufgabe 2: Leiten Sie aus den Personenobjekten eine oder mehrere Klasse(n) mit passenden Attributen ab und zeichnen diese in ein Klassendiagramm.
- Aufgabe 3: Versuchen Sie, aus den gefundenen Klassen eine oder mehrere gemeinsame Basisklasse(n) mit gemeinsamen Attributen zu entwickeln. Ergänzen Sie das Klassendiagramm und begründen das Design.

Die Lösungen zu den Aufgaben finden Sie in Kapitel 25, »Lösungen«, ab Seite 586.

## ■ 3.17 Literatur

Wikipedia: Alan Kay; [https://de.wikipedia.org/wiki/Alan\\_Kay](https://de.wikipedia.org/wiki/Alan_Kay)

# Stichwortverzeichnis

## A

Abfragemethode 31  
Abfragemethoden 280  
Abgeleitete Klasse 35  
ableiten 197  
Abschnittsbezogene Kommentare 402  
abstract 97  
Abstract Windowing Toolkit 480  
Abstrakte Klasse 202, 631  
Abstrakte Methode 631  
Abstraktion 32  
Aggregation 40  
Aktivitäten 416  
Algorithmen 535  
– entwickeln 536  
– verwenden 547  
Algorithmenarten 537  
Analyse 416  
Änderungsmethoden 31, 283  
Anforderungsaufnahme 416  
Annotation 195  
Anonyme Klassen 195  
Anweisung 139  
Applets 12, 493  
Arbeitsbereich 79  
Arithmetische Operatoren 298  
Arrays 247  
Assembler-Sprache 4  
assert 97  
Assoziation 39  
Attribut 28  
Aufzählungstyp 226  
Automatische Softwareaktualisierung  
– Einführung 80  
AWT 480

## B

Basisklassen 34, 463  
Betriebsphase 414  
Beziehung 39  
– ohne Vererbung 39  
Binärsystem 610  
Binärzahlen 610  
Bit 613  
Bitweise Operatoren 312  
Block 146  
Blockkommentare 403  
boolean 97, 170  
Border Layout 484  
break 97  
Build-System 77  
Byte 613  
byte 97, 164

## C

C/C++ 403  
case 97  
Cast-Operator 315, 521  
catch 97  
char 97, 170  
class 97  
Compiler 7, 77, 428  
Compilieren 428  
const 97  
Container 479  
continue 97  
Coprozessor 162

## D

Dateien schreiben 475, 476  
Debugger 78  
default 97, 506, 507  
Deklaration 141, 190  
Design 416

Designfehler 43  
 Designregel 46  
 Destruktor 30, 277  
 Dezimalsystem 609  
 Differenz-Operator 301  
 Digitalsystem 610  
 Digitalzahlen 610  
 Divisionsoperator 302  
 do 97  
 Dokumentationskommentare 402, 403  
 Doppelwort 613  
 Do-Schleife 352  
 double 97, 169  
 Dualsystem 610  
 Dynamische Polymorphie 45

**E**

Eclipse-Entwicklungsumgebung 54  
 Eclipse IDE 54  
 Eclipse-Plug-ins 78  
 Eclipse-Projektverwaltung 71  
 Editor 7, 72  
 else 97  
 Enterprise JavaBeans 496, 498  
 Entity Beans 497  
 Entwicklungsprozesse 413  
 Entwicklungsumgebung 7  
 enum 97  
 Ereignisbehandlung 481  
 Ereignissteuerung 493  
 Event-Handling 481  
 Exemplar 631  
 exports 97  
 extends 97, 197

**F**

false 97  
 Felder 247  
 Festkommazahl 164  
 FileReader 475  
 FileWriter 476  
 final 97, 197, 201  
 finalize 277  
 finally 97  
 float 97, 168  
 for 97  
 For-Schleife, einfach 353  
 For-Schleife, erweitert 354  
 Fragezeichenoperator 313  
 Funktion 278

**G**

Ganzzahl 164  
 Garbage Collector 449  
 GByte 613  
 Genauigkeit 162  
 Generalisierung 41, 631  
 Generics 207  
 Generische Klasse 207  
 Getter-Methoden 280  
 Glossar 631  
 goto 97  
 GridBag Layout 486

**H**

Hexadezimalsystem 612  
 Hilfesystem  
 – Einführung 81  
 Hochsprachen 6  
 Home Interface 498  
 HotJava 12  
 HotSwap 78

**I**

if 97  
 If-Verzweigung 328  
 implements 97  
 import 97  
 Import 364  
 Importanweisung 364  
 Innere Klasse 192  
 instanceof 97  
 Instantiierung 631  
 Instanzen 190, 631  
 Instanziieren 190  
 Instanziierung 631  
 int 97, 166  
 interface 97  
 Interfaces 204  
 Intro  
 – Einführung 81

**J**

Java 2 Micro Edition 498  
 JavaBean 493  
 Java Database Connectivity 493  
 Java-Editor 72  
 Java Enterprise Edition 496  
 Java-Klassenbibliotheken 459  
 Java-Laufzeitumgebung 441  
 Java ME 498  
 Java Micro Edition 498  
 Java Native Interface 494

Java-Schlüsselwörter 96  
Java-Standardbibliothek 462  
JDBC 493  
Jigsaw 14  
JNI 494  
JRE 441  
JVM 447  
JVM-Konfiguration 455

## K

Kapselung 37, 268, 506  
Kapselungsstärke 506  
KByte 613  
Kennung 29  
Klasse 25, 28, 207, 364, 631  
Klassenattribut 631  
Klassenbibliotheken 459  
Klassenimport 364  
Klassen kopieren 337  
Klassenmethoden 266, 632  
Klassenoperation 632  
Klassenvariable 124, 632  
Kompilieren 428  
Komposition 40  
Konfigurationsdateien 477  
Konkrete Klasse 189, 632  
Konstanten 29, 126  
Konstruktionsphase 414  
Konstruktor 30, 272  
– ohne Parameter 273

## L

Laufzeitumgebung 8, 441  
Layout-Manager 483  
Logische Operatoren 310, 615  
Logisches Und 310  
Lokale Klasse 194  
long 97, 167

## M

Maschinensprache 4, 6  
MByte 613  
Mehrfachvererbung 36  
Message Driven Beans 497  
Methode 30, 263, 631, 632  
Methodenaufruf 151  
Methodenimplementierung 269  
Methodenrumpf 269  
Modell 44  
Modellierung 44  
module 97  
Modulo-Operator 303

## N

Namensraum 366  
native 97  
Negation 310  
Neues Paket erzeugen 373  
new 97, 190  
New-Operator 314  
Nibble 613  
Nicht-Funktion 616  
Nicht-Operator 310  
null 97  
NullPointerException 619, 621

## O

Oak 12  
Object 463  
Objekt 25, 27, 632  
Objekte erzeugen 190  
Objektvariable 121, 122, 632  
Oder-Funktion 616  
Oder-Verknüpfung 311  
OOA/OOD 44  
OpenJDK 13  
Operation 32, 278  
Operator 297  
Outline 102

## P

Package 364, 366, 368  
package 97  
Paket 364, 368  
Parameter 120  
Persistenz 44  
Perspective 68  
Perspektive 68, 102  
Planungsphase 414  
Polymorphie 44, 526  
Postdekrement 305  
Postinkrement 304  
Prädekrement 305  
Präinkrement-Operator 303  
private 97, 506, 507  
Produkt 302  
Programmieren 3  
Projektverwaltung 7  
Properties 477  
protected 97, 506, 507  
public 97, 506, 507

## Q

Quotient-Operator 302

**R**

Rechnerunendlich 162  
Refactoring 44  
Remote Interface 498  
Remote Method Invocation 495  
requires 97  
return 97  
RMI 495  
Runtime 472

**S**

Schleifen 349  
Schleifenarten 350  
Schlüsselwörter 96  
Schnittstelle 204  
Sedezimalsystem 612  
Session Beans 497  
Setter-Methoden 283  
short 97, 165  
Sichtbarkeit 37, 268, 506  
Softwareaktualisierung  
– Einführung 80  
Sortieren 538, 547  
Speicher freigeben 449  
Standardkonstruktor 272  
Stateful Session Beans 497  
Stateless Session Beans 497  
static 97, 266, 632  
Statische Polymorphie 45  
strictfp 97  
String 465  
StringBuffer 469  
Subpaket erzeugen 373  
Summe 300  
Sun Microsystems 12  
super 97, 199, 632  
Superklasse Object 463  
Swing 481, 489  
Swing-Programme 555  
switch 97  
Switch-Anweisung 331  
Switch-Verzweigung 331  
synchronized 97  
System 470

**T**

TByte 613  
Test 416  
this 97, 199, 632  
Threads 473  
throws 97  
transient 97  
true 97

try 97  
Typkonvertierung 315, 521  
Typverletzung 207

**U**

Überladen von Methoden 526  
Überprüfung auf Gleichheit 307  
Überschreiben verhindern 531  
Überschreiben von Methoden 528  
Übersetzen 428  
Umstrukturierung 44  
Und-Operator 310  
Unicode 614  
Unterpaket erzeugen 373

**V**

var 97  
Variablenaufruf 150  
Vererbung 33, 197, 631  
Vergleich auf größer 309  
Vergleich auf größer oder gleich 309  
Vergleich auf kleiner 308  
Vergleich auf kleiner oder gleich 308  
Vergleich auf Ungleichheit 307  
Vergleichender Operator 306  
Verzweigungen 327, 328  
Virtuelle Maschine 447  
void 97  
volatile 97  
Vorzeichen 162, 299

**W**

Wahrheitswerte 170, 310, 615  
Werkzeug 417  
Wertebereich 162  
Wertzuweisung 146  
while 97  
While-Schleife 351  
Willkommenseite  
– Einführung 81  
Workbench 68  
Workbench-Fenster 68  
Wort 613  
Wrapper-Klassen 467, 469

**Z**

Zahlensysteme 609  
Zeichen 170  
Zeilenbezogene Kommentare 402  
Zeilenkommentare 402  
Zustand 29  
Zuweisung 143  
Zuweisungsoperatoren 312