

dirk LOUIS
peter MÜLLER



2. Auflage

Java

EINE EINFÜHRUNG IN
DIE PROGRAMMIERUNG

HANSER



Im Internet: Alle Beispiele und Setup-
Dateien für Java 9

Bleiben Sie auf dem Laufenden!



Unser **Computerbuch-Newsletter** informiert Sie monatlich über neue Bücher und Termine. Profitieren Sie auch von Gewinnspielen und exklusiven Leseproben. Gleich anmelden unter



www.hanser-fachbuch.de/newsletter



Hanser Update ist der IT-Blog des Hanser Verlags mit Beiträgen und Praxistipps von unseren Autoren rund um die Themen Online Marketing, Webentwicklung, Programmierung, Softwareentwicklung sowie IT- und Projektmanagement. Lesen Sie mit und abonnieren Sie unsere News unter



www.hanser-fachbuch.de/update



Dirk Louis
Peter Müller

Java

Eine Einführung
in die Programmierung

2. Auflage

HANSER

Die Autoren:

Dirk Louis, Saarbrücken, autoren@carpelibrum.de

Peter Müller, Saarbrücken, leserfragen@gmx.de

Alle in diesem Buch enthaltenen Informationen, Verfahren und Darstellungen wurden nach bestem Wissen zusammengestellt und mit Sorgfalt getestet. Dennoch sind Fehler nicht ganz auszuschließen. Aus diesem Grund sind die im vorliegenden Buch enthaltenen Informationen mit keiner Verpflichtung oder Garantie irgendeiner Art verbunden. Autoren und Verlag übernehmen infolgedessen keine juristische Verantwortung und werden keine daraus folgende oder sonstige Haftung übernehmen, die auf irgendeine Art aus der Benutzung dieser Informationen – oder Teilen davon – entsteht.

Ebenso übernehmen Autoren und Verlag keine Gewähr dafür, dass beschriebene Verfahren usw. frei von Schutzrechten Dritter sind. Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Buch berechtigt deshalb auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften.

Bibliografische Information der Deutschen Nationalbibliothek:

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

Dieses Werk ist urheberrechtlich geschützt.

Alle Rechte, auch die der Übersetzung, des Nachdruckes und der Vervielfältigung des Buches, oder Teilen daraus, vorbehalten. Kein Teil des Werkes darf ohne schriftliche Genehmigung des Verlages in irgendeiner Form (Fotokopie, Mikrofilm oder ein anderes Verfahren) – auch nicht für Zwecke der Unterrichtsgestaltung – reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

© 2018 Carl Hanser Verlag München, www.hanser-fachbuch.de

Lektorat: Brigitte Bauer-Schiewek

Copy editing: Petra Kienle, Fürstfeldbruck

Umschlagdesign: Marc Müller-Bremer, München, www.rebranding.de

Umschlagrealisation: Stephan Rönigk

Layout: Kösel Media GmbH, Krugzell

Druck und Bindung: Hubert & Co. GmbH & Co. KG BuchPartner, Göttingen

Printed in Germany

Print-ISBN: 978-3-446-45194-0

E-Book-ISBN: 978-3-446-45362-3

Inhalt

Vorwort	XI
1 Bevor es losgeht	1
1.1 Was ist Java? -Teil I	1
1.2 Was ist ein Programm?	3
1.3 Wie werden Programme erstellt?	5
1.4 Von Compilern und Interpretern	5
1.5 Was ist Java? - Teil II	6
1.6 Vorbereitung zum Programmieren	9
2 Der erste Kontakt	15
2.1 Die erste Java-Anwendung	15
2.2 Zusammenfassung	22
2.3 Fragen und Antworten	22
2.4 Übungen	22
3 Von Daten, Operatoren und Objekten	23
3.1 Variablen und Anweisungen	23
3.2 Operatoren	31
3.3 Typumwandlung	34
3.4 Objekte und Klassen	37
3.5 Arrays	48
3.6 Vordefinierte Klassen und Pakete	50
3.7 Zusammenfassung	51
3.8 Fragen und Antworten	52
3.9 Übungen	53
4 Programmfluss und Fehlererkennung mit Exceptions	55
4.1 Die Axiomatik des Programmablaufs	55
4.2 Modularisierung durch Klassen und Methoden	56

4.3	Kontrollstrukturen	66
4.4	Fehlerbehandlung durch Exceptions	77
4.5	Zusammenfassung	81
4.6	Fragen und Antworten	82
4.7	Übungen	83
5	Objektorientierte Programmierung mit Java	85
5.1	Vererbung	85
5.2	Methoden (Klassenfunktionen)	93
5.3	Variablen- und Methodensichtbarkeit	98
5.4	Innere Klassen	106
5.5	Mehrfachvererbung und Schnittstellen	107
5.6	Zusammenfassung	111
5.7	Fragen und Antworten	111
5.8	Übungen	113
6	Ein- und Ausgabe	115
6.1	Streams	115
6.2	Ausgaben auf den Bildschirm	116
6.3	Ausgabe in Dateien	121
6.4	Eingaben von Tastatur	124
6.5	Aus Dateien lesen	127
6.6	Ein wichtiger Punkt: korrekte Exception-Behandlung	129
6.7	Rund um Strings	131
6.8	Zusammenfassung	138
6.9	Fragen und Antworten	138
6.10	Übungen	139
7	Collections und weitere nützliche Klassen	141
7.1	Zufallszahlen erzeugen	141
7.2	Zeit- und Datumsangaben	143
7.3	Zeichenfolgen zerlegen	146
7.4	Komplexe Datenstrukturen (Collections)	147
7.5	Algorithmen	157
7.6	Zusammenfassung	159
7.7	Fragen und Antworten	159
7.8	Übungen	160
8	Grundlagen der GUI-Programmierung	161
8.1	Der GUI-Reiseführer	162
8.2	Aufbau einer GUI-Anwendung	163

8.3	Das Ereignis-Modell des AWT	168
8.3.1	java.awt.event importieren	171
8.3.2	Ereignislauscher definieren	171
8.3.3	Lauscher für Quelle registrieren	172
8.3.4	Adapter	173
8.3.5	Einige abschließende Anmerkungen	175
8.4	Chamäleon sein mit UIManager und Look&Feel	177
8.5	Ein umfangreicheres Beispiel	178
8.6	Zusammenfassung	179
8.7	Fragen und Antworten	180
8.8	Übungen	181
9	Grafik, Grafik, Grafik	183
9.1	Das Arbeitsmaterial des Künstlers	183
9.2	Erweitertes Layout mit Panel-Containern	192
9.3	Kreise, Rechtecke und Scheiben	194
9.4	Freihandlinien	198
9.5	Noch mehr Grafik mit Java2D	201
9.6	Zusammenfassung	205
9.7	Fragen und Antworten	206
9.8	Übungen	207
10	Bilder, Bilder, Bilder	209
10.1	Der Bildbetrachter	209
10.2	Dateien öffnen und speichern: die Klasse JFileChooser	214
10.3	Laden und Anzeigen von Bildern	216
10.4	Zusammenfassung	219
10.5	Fragen und Antworten	220
10.6	Übungen	220
11	Text, Text, Text	221
11.1	Ein Texteditor	221
11.2	Umgang mit Text: JTextField, JTextArea und JTextPane	222
11.3	Kombinationsfelder	228
11.4	Eigene Dialoge	231
11.5	Nach Textstellen suchen	235
11.6	Unterstützung der Zwischenablage	237
11.7	Drucken	239
11.8	Zusammenfassung	241
11.9	Fragen und Antworten	241
11.10	Übungen	242

12	Menüs und andere Oberflächenelemente	243
12.1	Die Komponentenhierarchie	243
12.2	Die Basisklasse Component	244
12.3	Statische Textfelder (JLabel)	246
12.4	Schaltflächen (JButton)	248
12.5	Eingabefelder (JTextField und JTextArea)	249
12.6	Optionen (JCheckBox, JRadioButton)	252
12.7	Listen- und Kombinationsfelder (JList und JComboBox)	254
12.8	Bildlaufleisten (JScrollBar)	256
12.9	Menüleisten (JMenuBar)	258
12.10	Zusammenfassung	260
12.11	Fragen und Antworten	261
12.12	Übungen	261
13	Threads und Animation	265
13.1	Multithreading mit Java	265
13.2	Eigene Threads erzeugen: die Klasse Thread	268
13.3	Eigene Threads erzeugen: die Runnable-Schnittstelle	271
13.4	Wissenswertes rund um Threads	273
13.5	Threads und Animation I	275
13.6	Threads und Animation II	279
13.6.1	SwingWorker	280
13.7	Zusammenfassung	284
13.8	Fragen und Antworten	285
13.9	Übungen	285
14	Sound	287
14.1	Was ist eine URL?	287
14.2	Sounddateien abspielen	288
14.3	Wiedergabe von MP3	289
14.4	Tonerzeugung mit MIDI	290
14.4.1	Abspielen einer MIDI-Datei	290
14.4.2	Selber Musik machen	291
14.5	Zusammenfassung	292
14.6	Fragen und Antworten	293
14.7	Übungen	293
15	Die Datenbankschnittstelle JDBC	295
15.1	Datenbanken-ABC	295
15.2	Die JDBC-Schnittstelle	297
15.3	Vorbereitung für JavaDB	298

15.4	Zugriff auf eine Datenbank	298
15.4.1	Verbindungsaufbau	299
15.4.2	Lese- und Schreiboperationen durchführen	299
15.4.3	Verbindung schließen	301
15.5	Zusammenfassung	304
15.6	Fragen und Antworten	305
15.7	Übungen	305
16	Was wir noch erwähnen wollten	307
16.1	Aufzählungen (enum)	307
16.1.1	Definition	308
16.1.2	Variablen definieren	308
16.1.3	Aufzählungskonstanten vergleichen	308
16.1.4	Aufzählungen und switch	309
16.1.5	Aufzählungen und for	309
16.2	Lambda-Ausdrücke	310
16.3	Java Generics	311
16.3.1	Einleitung	311
16.3.2	Syntax	312
16.3.3	Eingeschränkte Platzhalter	314
16.3.4	Parameter und Variablen von generischen Typen	314
16.4	Jar-Archive	315
16.5	Module	316
16.6	Debuggen	318
16.6.1	Grundsätzliches Vorgehen	319
16.7	Anwendungen weitergeben	320
16.7.1	Ohne JRE geht es nicht	320
16.7.2	Java-Anwendungen ausführen: von .class bis .exe	322
Anhang A:	Lösungen	325
Anhang B:	Installation des JDK	341
B.1	Installation	342
B.2	Anpassen des Systems	344
B.2.1	Erweiterung des Systempfads	344
B.2.2	Installation testen	346
B.2.3	Setzen des Klassenpfads	347
B.3	Die Java-Dokumentation	348
B.4	Wo Sie weitere Hilfe finden	349
Anhang C:	Schlüsselwörter	351

Anhang D: Java-Klassenübersicht	353
D.1 java.io	353
D.2 java.lang	354
D.3 java.applet	354
D.4 java.awt	355
D.5 java.awt.event	356
D.6 java.awt.geom	357
D.7 java.net	357
D.8 java.sql	358
D.9 javax.sound.midi	358
D.10 javax.swing	359
D.11 java.util	360
Anhang E: Literatur und Adressen	361
E.1 Bücher	361
E.2 Zeitschriften	362
E.3 Ressourcen im Internet	363
Anhang F: Das Material zum Buch	365
Index	367

Vorwort

Dieses Buch soll Sie auf leicht verständliche und gleichsam unterhaltsame Weise in die Programmierung mit Java einführen. Vorhandene Programmierkenntnisse können von Vorteil sein, werden aber nicht vorausgesetzt. Schritt für Schritt werden Sie sich in Java einarbeiten und erfahren, wie Sie die Mächtigkeit der Sprache für Ihre Zwecke nutzen können.

Sie werden Ihre ersten Java-Anwendungen schreiben, sich mit der objektorientierten Programmierung vertraut machen und erfahren, wie man in Java Anwendungen mit grafischen Benutzeroberflächen programmiert. In den letzten Kapiteln des Buchs wenden wir uns dann noch diversen Fortgeschrittenenthemen zu, wie der Implementierung von Threads und Animationen oder der Datenbankunterstützung.

Am Ende eines jeden Kapitels finden Sie eine Reihe von Testfragen und eine Zusammenfassung des behandelten Stoffs sowie einige Übungsaufgaben, die zur Vertiefung und Wiederholung des Stoffs dienen, vor allem aber auch die Lust am Programmieren anregen sollen.

Programmierkenntnisse sind wie erwähnt nicht erforderlich, aber Sie sollten auch nicht zu den Zeitgenossen gehören, die bisher noch jedem Computer erfolgreich aus dem Weg gegangen sind. Spaß am Programmieren können wir Ihnen nur dann vermitteln, wenn Sie selbst ein wenig guten Willen und Ausdauer mitbringen. Gute Laune, eine Portion Neugier, dieses Buch – was brauchen Sie noch?

Um mit Java programmieren zu können, benötigen Sie ein entsprechendes Entwicklungspaket, das sogenannte JDK (Java Development Kit). Optional können Sie zusätzlich zu dem JDK eine integrierte Entwicklungsumgebung wie z. B. NetBeans oder die Open-Source-Software Eclipse verwenden.

Für Java-Einsteiger ist der Einsatz einer integrierten Entwicklungsumgebung allerdings häufig recht verwirrend und lenkt vom eigentlichen Primärziel, die Programmiersprache Java zu lernen, unnötig ab. Schlimmer noch: Die meisten Entwicklungsumgebungen sind für fortgeschrittene Programmierer konzipiert und erleichtern deren tägliche Programmierarbeit, indem sie komplexe Arbeitsschritte automatisieren, vordefinierte Codegerüste anbieten, eigenständig Code erzeugen. Für Anfänger ist dies ein Desaster! Nehmen wir nur einmal das Codegerüst einer einfachen Konsolenanwendung, das in Java aus ungefähr vier bis sieben Zeilen Code besteht. Wenn Ihnen dieses Codegerüst stets von Ihrer Entwicklungsumgebung fertig vorgelegt wird, werden Sie sich kaum die Mühe machen, es je selbst einmal abzutippen. Sie werden es sich vielleicht anschauen und versuchen, es zu verstehen, aber Sie werden es nie wirklich verinnerlichen. Wenn Sie dann später einmal an einem Rechner

arbeiten müssen, auf dem keine Entwicklungsumgebung installiert ist, werden Sie mit Schrecken feststellen, dass es Ihnen unmöglich ist, das Grundgerüst aus dem Kopf nachzustellen. Nun, ganz so schlimm wird es vielleicht nicht kommen, aber der Punkt ist, dass die Annehmlichkeiten der Entwicklungsumgebungen den Anfänger schnell dazu verführen, sich mit zentralen Techniken und Prinzipien der Java-Programmierung nur oberflächlich auseinanderzusetzen.

Aus diesem Grund legen wir in diesem Buch Wert auf Handarbeit mit elementarsten Mitteln: Wir setzen unsere Quelltexte in einem einfachen Texteditor auf, wandeln die Quelltexte mithilfe des Java-Compilers *javac* aus dem JDK in ausführbare Programme um und führen diese dann mit dem Java-Interpreter *java* (ebenfalls im JDK enthalten) aus. Wie Sie dabei im Einzelnen vorgehen und was Sie beachten müssen, erfahren Sie in den einleitenden Kapiteln und im Anhang dieses Buchs.

Benötigte Software und Beispielsammlung

Die Java-Entwicklungsumgebung und die Beispielsammlung zu diesem Buch finden Sie zum Download im Internet. Die entsprechenden Links finden Sie im Anhang F, Hinweise zur Installation der Entwicklungsumgebung in Anhang B.

www.carpelibrum.de

Falls Sie während der Buchlektüre auf Probleme oder gar auf inhaltliche Fehler stoßen, sollten Sie nicht zögern, uns eine E-Mail unter Angabe von Buchtitel und Auflage zu senden. Allerdings schauen Sie bitte zuerst auf unserer Buchseite www.carpelibrum.de nach, ob sich nicht dort schon eine Antwort findet. Neben Aktualisierungen, Fehlerkorrekturen und Antworten auf typische Fragen finden Sie dort auch Hinweise auf weitere Bücher rund ums Thema Programmieren.

Viel Erfolg mit Java wünschen Ihnen

Dirk Louis (autoren@carpelibrum.de)

Peter Müller (leserfragen@gmx.de)

Saarbrücken, im Frühjahr 2018

1

Bevor es losgeht

Ich weiß, ich weiß – Sie sitzen vor Ihrem Bildschirm, haben bereits Ihre Java-Entwicklungs-umgebung installiert, brennen darauf, Ihr erstes Java-Programm zu schreiben, und sind einigermaßen ungehalten, sich erst noch durch etliche Seiten theoretischer Ausführungen quälen zu müssen. Müssen Sie nicht! Wenn Sie Ihre Entwicklungsumgebung schon eingerichtet haben und mit der Programmerstellung prinzipiell vertraut sind, überspringen Sie dieses Kapitel einfach. Nur wenn Sie ein absoluter Neuling in der Programmierung sind oder von Java nicht viel mehr wissen, als dass es eine Programmiersprache ist, sollten Sie dieses Kapitel unbedingt vorab durchlesen. Alle anderen können nach Bedarf auch noch später hierher zurückkehren.

■ 1.1 Was ist Java? –Teil I

Java ist heute eine der führenden Programmiersprachen, vielleicht die wichtigste Programmiersprache überhaupt. Programmierer weltweit schätzen Java für seine Robustheit, seine Vielseitigkeit, die problemlose Portierbarkeit seiner Anwendungen und, und, und.



Portierung bedeutet, dass ein Programm von einem Rechner auf einen anderen Rechner verschoben und ausgeführt wird.

Entstanden ist Java 1993 als Forschungsprojekt der Firma Sun, wobei schon auf diverse Vorarbeiten zurückgegriffen werden konnte. Der konkrete Anlass war der einsetzende Boom des World Wide Web, das nach einer geeigneten Programmiersprache verlangte. Java ist vor diesem Hintergrund zu betrachten und quasi ideal für den Einsatz im Internet – sei es, dass man seine Programme über das Internet vertreiben möchte, sei es, dass man Programme für Webseiten schreiben möchte, oder sei es, dass man über das Internet verteilte Anwendungen implementieren möchte.

Wie steht es in diesem Zusammenhang mit der Verwandtschaft von Java zu den anderen Programmiersprachen? Man entwickelt schließlich keine neue Programmiersprache, ohne die eigenen Erfahrungen mit den etablierten Programmiersprachen einfließen zu lassen.

Nun, C++-Programmierer wird es freuen zu hören, dass Java stark an C++ angelehnt ist. Die Gründe hierfür sind zweifellos in der Objektorientiertheit, der Schnelligkeit und der Leistungsfähigkeit von C++ zu suchen, aber natürlich auch in der traditionellen Bedeutung dieser Sprache.

Allerdings hat Java viel unnötigen Ballast, den C++ mit sich schleppt, abgeworfen und ist dadurch wesentlich einfacher zu erlernen und zu programmieren. Diese Entschlackung dient nicht nur der Entlastung des Programmierers, sondern soll vor allem auch die Entwicklung „sicherer“ Programme gewährleisten. Natürlich liegt die Verantwortung für die Sicherheit der Anwendungen deswegen letztendlich immer noch beim Programmierer. Je komplizierter und undurchsichtiger die Konzepte einer Sprache aber sind, umso wahrscheinlicher ist es, dass der Programmierer unbeabsichtigt Fehler einbaut. In Java hat man dies erkannt und beispielsweise die gesamte Zeigerprogrammierung und die dynamische Speicherverwaltung aus den Händen des Programmierers genommen und Compiler und Interpreter übertragen.



Und wie steht es mit C#? Die Sprache C# ist im Grunde nichts anderes als eine Trotzreaktion auf Java. Microsoft wollte nämlich ursprünglich eine eigene Java-Variante etablieren, was dem Konzern aber gerichtlich verboten wurde. Danach konzipierte man eine ganze neue Sprache, eben C#, die aber eine verblüffende Ähnlichkeit mit Java aufweist. Umsteigern von C# wird also vieles bekannt vorkommen.

Falls Sie schon C++ beherrschen – die folgenden Konzepte gibt es in Java *nicht*:

1. Zeiger (die dynamische Speicherverwaltung wird intern vorgenommen)
2. Funktionen (statt alleinstehender Funktionen gibt es nur noch Methoden (Elementfunktionen) von Klassen)
3. Strukturen und Unions
4. Arrays und Zeichenfolgen gibt es nur als Objekte
5. Typendefinition (typedef)
6. Mehrfachvererbung (nur in gemäßigter Form)
7. Überladung von Operatoren

Java deshalb als Schmalspur-C++ zu bezeichnen, wäre aber völlig falsch. Von der Leistungsfähigkeit her steht Java C++ kaum in etwas nach. Betrachtet man obige Liste etwas genauer, lässt sich feststellen, dass viele Konzepte, die C++ von C übernommen hat, zugunsten einer konsequenteren objektorientierten Programmierung aufgegeben wurden (dies betrifft die Sprachelemente 2 bis 5, die alle im Klassenkonzept aufgegangen sind). Java ist daher mittlerweile die Standardprogrammiersprache an allen US-amerikanischen Universitäten und auch an deutschen Universitäten allgegenwärtig.

Andererseits wurde auf bestimmte objektorientierte Konzepte (Punkte 6 und 7), die im Wesentlichen der Wiederverwertung objektorientierten Quellcodes dienen, aber für Einsteiger (und auch oft noch für Fortgeschrittene) manchmal schwierig zu handhaben sind, verzichtet. Was geblieben ist, ist eine relativ leicht zu erlernende, konsequent objektorientierte Sprache, die Ihnen einiges zu bieten hat:

- Objektorientiertheit,
- statische Typbindung, aber späte Methodenbindung,
- dynamische Speicherverwaltung und Garbage Collection,
- Multithreading,
- Exception-Behandlung.



Achtung!

Verwechseln Sie Java nicht mit JavaScript. JavaScript wurde von Netscape entwickelt und als Erweiterung des HTML-Standards implementiert. Die JavaScript-Syntax ist an Java angelehnt, doch damit hört die Verwandtschaft zu Java auch schon auf. Mit Java können Sie echte eigenständige Programme schreiben (das Thema dieses Buchs), Sie können Java-Servlets schreiben, die dynamische Webseiten erzeugen, und Sie können spezielle Java-Module erzeugen, sogenannte Applets, die in Webseiten eingebettet und von Browsern mit installiertem Java-Plug-in ausgeführt werden können (früher eine Sensation, heute kaum noch von Bedeutung). Dagegen dient JavaScript-Code allein der Dynamisierung von Webseiten, weswegen der Code auch direkt in den HTML-Code eingefügt und vom Browser interpretiert wird (sofern die JavaScript-Unterstützung nicht ausgeschaltet wurde).

■ 1.2 Was ist ein Programm?

Prinzipiell sind Programme nichts anderes als eine Folge von Befehlen, die an einen Computer gerichtet sind und von diesem befolgt werden. Im Grunde genommen funktionieren Programme also genauso wie Kochrezepte: Sie als Programmierer sind der Koch, der das Buch schreibt. Jedes Kochrezept entspricht einem Programm und der Computer, der Ihre Programme ausführt, ist der Leser.

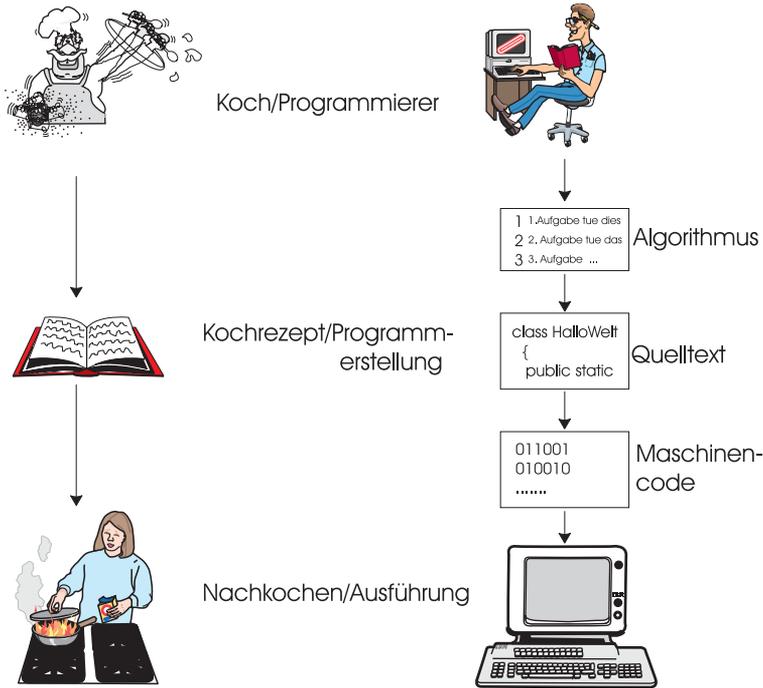


Bild 1.1 Analogie zwischen Programmen und Kochrezepten

Leider ist die Realität wie üblich etwas komplizierter als das Modell. Im Falle des Kochrezepts können wir einfach davon ausgehen, dass Schreiber und Leser die gleiche Sprache sprechen. Im Falle des Programmierers und des Computers ist dies natürlich nicht der Fall, denn Sie als Programmierer sprechen an sich Deutsch und der Computer spricht ... ja, welche Sprache versteht eigentlich ein Computer?

Ich wünschte, Sie hätten diese Frage nicht gestellt, denn die Antwort ist äußerst unerfreulich. Der Computer, in diesem Fall sollte man genauer von dem Prozessor des Computers sprechen, versteht nur einen ganz begrenzten Satz elementarer Befehle – den sogenannten Maschinencode, der zu allem Unglück noch binär codiert ist und daher als eine Folge von Nullen und Einsen vorliegt. Können Sie sich vorstellen, Ihre Programme als Folge von Nullen und Einsen zu schreiben? Wahrscheinlich genauso wenig, wie Ihr Computer in der Lage ist, Deutsch zu lernen. Wir haben also ein echtes Verständigungsproblem. Um dieses zu lösen, müssen Sie – als der Intelligentere – dem Computer schon etwas entgegenkommen.

Kehren wir noch einmal zu unserem Kochbuch zurück und stellen Sie sich vor, ein Chinese würde ein Kochbuch schreiben, das auf dem deutschen Buchmarkt erscheinen soll. Zwar findet der Chinese keinen geeigneten Übersetzer, der das Buch ordentlich vom Chinesischen ins Deutsche übersetzen könnte, aber er erinnert sich seiner Englischkenntnisse, die für ein Kochbuch absolut ausreichend sein sollten. Er schreibt also sein Buch in Englisch und lässt es dann von einem Übersetzer ins Deutsche übertragen. Gleiches geschieht auch bei der Programmierung. Anstatt Ihre Programme in Deutsch aufzusetzen, bedienen Sie sich einer Programmiersprache (wie Java, C, Pascal, Basic etc.), für die es einen passenden

Übersetzer gibt (in diesem Fall auch Compiler genannt), der Ihre Anweisungen in Maschinencode umwandeln kann.

Ich denke, das ist jetzt schon etwas klarer. Was aber genau ist jetzt das Programm? Die noch in Deutsch formulierten Befehle? Die in Java formulierten Befehle? Oder die binär codierten Maschinenanweisungen? Im weitesten Sinne können Sie in allen drei Fällen von Ihrem Programm reden. Wenn Sie es dagegen genau nehmen wollen, bezeichnen Sie die noch in Ihrer Sprache aufgesetzte Befehlsfolge als Algorithmus, die in Java formulierte Version des Algorithmus als Quelltext Ihres Programms und erst den vom Compiler erzeugten Maschinencode als Ihr Programm.

Mehr oder weniger unabsichtlich sind wir damit bereits in die Programmerstellung abgeglitten, die im nächsten Abschnitt noch einmal zusammengefasst wird.

■ 1.3 Wie werden Programme erstellt?

Die Entwicklung von Computerprogrammen läuft unabhängig von der verwendeten Sprache üblicherweise nach dem folgenden Muster ab:

1. Man hat ein Problem, eine Idee, eine Aufgabe, zu deren Lösung man einen Computer einsetzen möchte.
2. Als Nächstes wird die Aufgabe als Algorithmus, also als eine Folge von Befehlen formuliert. Größere Probleme werden dabei in Teilaufgaben und Teilaspekte aufgeteilt. (Ob der Algorithmus tatsächlich auf dem Papier oder nur im Kopf des Programmierers entwickelt wird, hängt von der Komplexität der Aufgabe und der Genialität des Programmierers ab.)
3. Der Algorithmus wird in für den Computer verständliche Anweisungen einer Programmiersprache umgesetzt. Dies ergibt den sogenannten Quelltext oder Quellcode.
4. Dieser Quelltext muss dann durch ein spezielles Programm, den Compiler, in Maschinenanweisungen übersetzt werden, die das eigentliche Herz des Computers – der Prozessor – versteht und ausführen kann.
5. Das ausführbare Programm wird gestartet, das heißt in den Hauptspeicher geladen und vom Prozessor ausgeführt.

■ 1.4 Von Compilern und Interpretern

Bei einigen Programmiersprachen fallen die Schritte 4 und 5 zusammen. Es wird also nicht das ganze Programm erst übersetzt (kompiliert) und dann bei Bedarf ausgeführt. Stattdessen wird bei der Ausführung des Programms der Quelltext Zeile für Zeile eingelesen, übersetzt und ausgeführt. In diesem Fall spricht man von Interpreter-Sprachen, weil das Programm nicht als ausführbare Datei, sondern bloß als Quelltext vorliegt, der nur mithilfe eines speziellen Programms (des Interpreters), welches die zeilenweise Übersetzung wäh-

rend des Programmlaufs übernimmt, ausgeführt werden kann. In dem Beispiel aus Abschnitt 1.2 würde dies bedeuten, dass der chinesische Koch seine Kochkünste nicht in Buchform, sondern als Hörkassette herausgegeben hat und Sie mit einem Dolmetscher (Interpreter) an der Seite diese Kassette abspielen und die Rezepte nachkochen.

Abgesehen davon, dass Sie wahrscheinlich niemals auf die Idee kommen werden, sich eine chinesische Kassette mit Kochrezepten zu kaufen, haften interpretierten Programmen zwei wesentliche Nachteile an:

- Da die Erzeugung des Maschinencodes erst während der Ausführung vorgenommen wird, dürfte klar sein, dass solche Programme wesentlich langsamer ablaufen als kompilierte Programme.
- Da diese Programme als Quelltext vertrieben werden, sind der nicht autorisierten Nutzung des Programmtextes und der für das Programm entwickelten Algorithmen Tür und Tor geöffnet.

Auf der anderen Seite haben diese Programme den Vorteil, dass sie sehr gut zu portieren sind, das heißt, die Übertragung von einem Computer auf einen anderen ist unproblematisch.



Bekannte Vertreter von Compiler-Sprachen sind C, C++ und Pascal. Das klassische Beispiel für eine interpretierte Programmiersprache ist Basic.

Jetzt fragen Sie sich sicherlich, wo Java einzuordnen ist.

■ 1.5 Was ist Java? – Teil II

Ist Java nun eine Compiler- oder eine Interpreter-Sprache? Nun, die Einteilung ist hier nicht ganz so klar. Man kann zwar vielerorts lesen, es sei eine Interpreter-Sprache, aber das wird der Lage nicht ganz gerecht, denn tatsächlich ist Java beides.

Java-Quellcode wird zunächst mit einem Compiler (er heißt *javac*) übersetzt, allerdings nicht in den Maschinencode des jeweiligen Prozessors, sondern in sogenannten Bytecode. Man kann sich diesen Bytecode als einen Maschinencode eines virtuellen Prozessors vorstellen, das heißt eines Prozessors, den es gar nicht gibt!

Damit der Bytecode nun von einem echten Prozessor ausgeführt werden kann, muss er während des Programmlaufs in dessen Maschinencode übersetzt werden, das heißt, ein Interpreter ist zum Ausführen von Java-Programmen notwendig, den man einfach wie die Sprache getauft hat, also *java*.

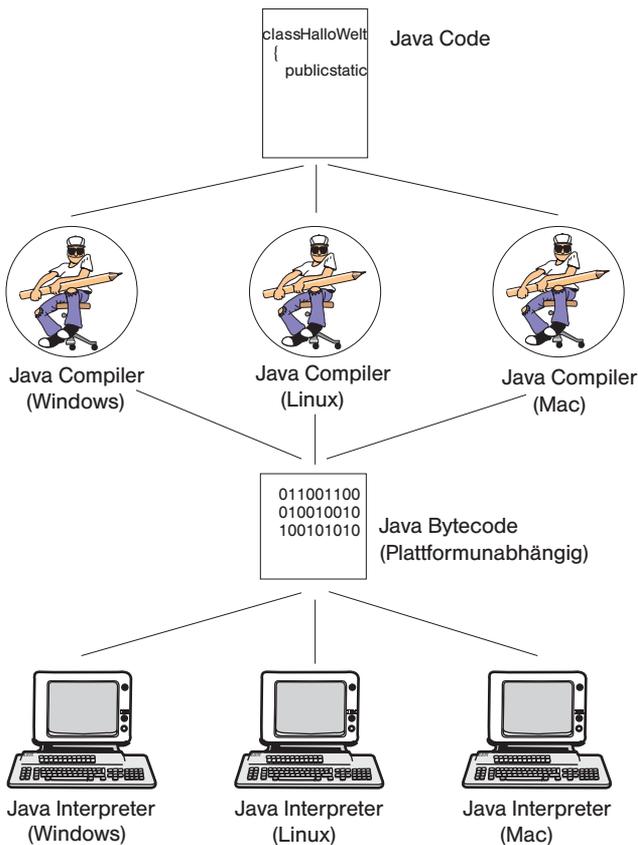


Bild 1.2 Erstellung von Java-Programmen

Warum diese seltsame Mischung aus Compiler- und Interpretersprache? Die Antwort liegt eigentlich auf der Hand. Es geht darum, die Vorteile beider Systeme miteinander zu verbinden:

- Der vorkompilierte Bytecode kann wesentlich schneller interpretiert werden.
- Die Übersetzung in Bytecode schützt Ihre Algorithmen vor unerwünschter Nachahmung.
- Der Bytecode ist plattformunabhängig. Gleichgültig, auf welchem Computer der Bytecode erstellt worden ist, er wird immer identisch aussehen. Es ist dadurch möglich, Bytecode von einem Rechner zum nächsten zu transferieren (zum Beispiel über das Internet) und dort ausführen zu lassen, ohne die *geringste* Änderung vornehmen zu müssen. (Wenn Sie schon in konventionellen Sprachen wie C programmiert haben und das Problem hatten, ein Programm zu entwerfen, das auf verschiedenen Betriebssystemen, Prozessoren und Compilern läuft, werden Sie diesen Vorteil von Java sicherlich zu schätzen wissen.)
- Der Interpreter hat die Möglichkeit, die Programmausführung zu überwachen und beispielsweise (absichtliche oder unabsichtliche) unkorrekte Speicherzugriffe oder Datenmanipulationen zu erkennen und direkt mit einer Fehlermeldung abzufangen. Dies ist insbesondere unter Sicherheitsaspekten interessant. Da Java für das Internet und somit

für den beliebigen Datenaustausch konzipiert ist, muss weitgehend sichergestellt sein, dass ein Java-Programm, das man mit seinem Browser durch einen mehr oder weniger achtlosen Klick startet, nicht irgendwelche üblen Sachen anstellt. Ein Interpreter, der ständig darauf achtet, was das Java-Programm gerade tun will, ist dafür ein geeigneter Ansatz.

Zusammenfassend lässt sich sagen, dass Java-Programme aufgrund des Compiler/Interpreter-Ansatzes hardwareunabhängig und somit portabel sind und gleichzeitig ein relativ hohes Maß an Sicherheit vor Manipulationen bieten. Diese Vorteile gehen jedoch zu Lasten der Geschwindigkeit. Interpretierte Java-Programme sind trotz Bytecode langsamer als vergleichbare C/C++-Programme. Dies sollte Sie aber nicht abschrecken. Moderne Prozessoren sind schnell genug und viele Programme verbringen die meiste Zeit sowieso mit Warten auf Benutzereingaben, sodass der Geschwindigkeitsnachteil gar nicht zum Tragen kommt. Zudem verwenden modernen Java-Interpreter spezielle Techniken, wie z. B. die sogenannte Just-In-Time-Compilation (JIT), die den Geschwindigkeitsnachteil minimal halten.¹



Die Java-Laufzeitumgebung (JRE)

Sie haben gelernt, dass auf einem Rechner, auf dem Java-Programme ausgeführt werden sollen, ein Java-Interpreter installiert sein muss (siehe auch Bild 1.2). Dies war insofern etwas vereinfacht dargestellt, als nicht nur ein einzelnes Interpreter-Programm, sondern eine ganze Kombination aus Tools und Bibliotheken benötigt werden – die sogenannte Java-Laufzeitumgebung, abgekürzt **JRE** („Java Runtime Environment“).

Wenn Sie also Ihre Java-Programme an Freunde weitergeben, sagen Sie ihnen, dass sie sich gegebenenfalls die JRE von der Site <http://www.oracle.com/technetwork/java/javase/downloads/index.html> herunterladen müssen. Ausgeführt werden die Programme dann durch Übergabe an den *java*-Interpreter, so wie Sie es gleich weiter unten lernen werden. Oder Sie erzeugen für Ihre Freunde eine direkt ausführbare *.jar*-, *.bat*- oder *.exe*-Datei, siehe Kapitel 16.

¹ Bei der Just-In-Time-Compilation übersetzt der Java-Interpreter *während der Programmausführung* Programmteile, die oft benötigt werden, in optimierten Maschinencode für den verwendeten Prozessor und hält diese Codeblöcke im Hauptspeicher, sodass bei Wiederverwendung während der aktuellen Programmausführung keine erneute Übersetzung notwendig ist. Zusätzlich werden häufig fortlaufend Hot-Spot-Analysen gemacht, d. h., sehr oft durchlaufene Code-Abschnitte (die „Hot-Spots“) werden parallel zur Programmausführung intensiv optimiert und dadurch noch schneller gemacht.

■ 1.6 Vorbereitung zum Programmieren

Bevor wir im nächsten Kapitel voll in die Programmierung einsteigen, sollten Sie Ihren Computer vorbereitet haben. Folgende Zutaten sind für die Erstellung von Java-Programmen notwendig:

■ die Java-Entwicklungswerkzeuge (JDK)

Das JDK steht unter <http://www.oracle.com/technetwork/java/javase/downloads/index.html> zum Download zur Verfügung (siehe Anhang F, Installationshinweise siehe Anhang B).

Das JDK² besteht aus einer Sammlung aller für die Programmierung erforderlichen Tools (Compiler, Interpreter, Debugger etc.) und den zu Java gehörenden Standardbibliotheken. Die JDK-Tools werden von der Konsole aus aufgerufen. Linux-Anwender öffnen hierzu ein Terminalfenster, Windows-Anwender die Eingabeaufforderung.

■ ein Texteditor

Für die Programmierung genügt ein einfacher Texteditor, wie er zur Grundausstattung jedes anständigen Betriebssystems gehört. Wichtig ist, dass der Editor unformatierten Text (also einfach nur die Buchstaben ohne Formatangaben wie fett, kursiv etc.) abspeichern kann.

Mögliche Optionen wären *Editor* (*notepad.exe*) oder *Wordpad* unter Microsoft Windows (Aufruf über die Suche der Windows-Startseite (Windows 8) bzw. des Windows-Startmenüs (Windows 7)) bzw. *vi*, *KWrite* oder *emacs* unter Linux/Unix. Natürlich können Sie auch Textverarbeitungssoftware wie *Microsoft Word* einsetzen, allerdings müssen Sie darauf achten, nur reinen unformatierten Text (ASCII bzw. ANSI, Dateiendung *.txt*) abzuspeichern.



Auch wenn konventionelle Texteditoren wie *Editor* oder *Wordpad* grundsätzlich vollkommen ausreichen, lassen sie doch verschiedene Extras vermissen, die gerade dem Programmierer die Arbeit sehr erleichtern können – wie z. B. die automatische Syntaxhervorhebung oder die Anzeige der Zeilennummern³. Wer auf diese Funktionen nicht verzichten möchte, muss zu einem Editor greifen, der sich auf die Quelltextverarbeitung spezialisiert hat. Empfehlenswerte Vertreter dieser Gattung sind z. B. der *emacs* für Linux (gehört in der Regel zum Lieferumfang) und der *Notepad++*-Editor für Windows (kann von <http://notepad-plus.sourceforge.net/de/site.htm> kostenlos heruntergeladen werden).

² Abkürzung für „Java SE Development Kit“

³ Der Compiler gibt Meldungen zu Syntaxfehlern immer unter Angabe der betroffenen Zeilennummer aus. Wenn Sie einen Editor mit Zeilennummerierung verwenden, müssen Sie nicht selbst zählen, um die verdächtige Zeile zu finden.

Einrichtung einer eigenen Entwicklungsumgebung

Als Erstes sollten Sie sich überlegen, wie Sie die Dateien Ihrer Programme auf der Festplatte verwalten wollen. Eine Möglichkeit wäre, unter einem eigenen übergeordneten Verzeichnis für die einzelnen Programme Unterverzeichnisse anzulegen. Zum Nachvollziehen der Beispiele in diesem Buch bietet es sich allerdings an, unter dem übergeordneten Verzeichnis Unterverzeichnisse für die Buchkapitel anzulegen.

Wenn Sie das übergeordnete Verzeichnis *Java* nennen, könnte Ihre Verzeichnisstruktur wie folgt aussehen:

C:\Java

C:\Java\Kap02

C:\Java\Kap03

C:\Java\Kap04

...

In den Verzeichnissen für die einzelnen Kapitel speichern Sie dann die Quelldateien der zugehörigen Beispiele. Für kleinere Beispiele können Sie die Quelldateien ruhig zusammen im Kapitelverzeichnis speichern. Für größere Beispiele, die aus mehreren Quelldateien bestehen, empfiehlt es sich, nochmals eigene Unterverzeichnisse für jedes Programm anzulegen. (Gleiches gilt, wenn zwei Programme gleichnamige Klassen definieren.)

1. Installieren Sie zuerst das *JDK*, siehe Anhang B.
2. Richten Sie danach auf Ihrem Rechner ein Verzeichnis *Java* ein.

Unter Windows können Sie das Verzeichnis direkt unter *C:* anlegen, also als *C:\Java*. (Unterverzeichnisse können im Windows Explorer, Aufruf mit **WinBef+E**, erzeugt werden.)

Unter Linux legen Sie das Verzeichnis unter Ihrem Home-Verzeichnis an.

3. Legen Sie ein Unterverzeichnis *Kap02* für Kapitel 2 an.

Jetzt sollten Sie Ihren Desktop noch so gestalten, dass Sie alle für die Programmierung typischen Aufgaben effizient erledigen können.

4. Lassen Sie den Windows Explorer (Ordnerfenster) aus Schritt 1 geöffnet. Sie brauchen ihn, um bei Bedarf weitere Kapitel- oder Programmverzeichnisse anzulegen. Außerdem können Sie aus dem Explorer heraus bestehende Quelldateien zur Ansicht oder Bearbeitung öffnen.
5. Rufen Sie den Texteditor Ihrer Wahl auf, mit dem Sie Ihren Quellcode als reinen Text (ASCII oder ANSI, kein Unicode) abspeichern können. Wenn Sie keinen spezialisierten Quelltexteditor besitzen (vgl. Anmerkung zu *Notepad++* weiter oben), verwenden Sie einfach einen der Editoren aus dem Lieferumfang Ihres Betriebssystems.

Unter Windows eignet sich beispielsweise wie gesagt der *Editor* (*notepad.exe*), den Sie über die Windows-Suche, Suchbegriff *notepad*, aufrufen können. (Alternativ finden Sie den Editor je nach Windows-Version in der App-Liste unter der Kategorie **Windows-Zubehör** bzw. im Startmenü unter **Alle Programme/Zubehör**.)

Unter Linux können Sie z. B. den *vi* oder *KWrite* verwenden.

6. Speichern Sie zur Probe im Verzeichnis *C:\Java\Kap02* eine Datei *HalloWelt.java*. Die Datei kann ruhig leer sein (oder einen beliebigen Text enthalten).

**Achtung!**

Achten Sie darauf, dass der Editor nicht die Dateierdung `.txt` an die gespeicherten Dateien anhängt (also aus `Dateiname.java` die `Dateiname.java.txt` macht). Nach Installation des JDK sollte dies an sich nicht passieren. Falls doch, haben Sie zwei Möglichkeiten. Die erste Lösung besteht darin, den kompletten Dateinamen, samt Dateierdung, in Anführungszeichen zu setzen: `"Dateiname.java"`. Die zweite Möglichkeit ist, die Dateierdung `.java` im Windows Explorer zu registrieren. Speichern Sie dazu nach Methode 1 eine Datei mit der Dateierdung `.java`. Wechseln Sie danach in den Windows Explorer und doppelklicken Sie auf die Datei. Ist die Dateierdung noch nicht registriert, erscheint jetzt ein Dialogfenster (oder eine Folge von Dialogfenstern), in denen Sie *Notepad* als gewünschtes Bearbeitungsprogramm auswählen und die Option setzen können, die dafür sorgt, dass dieser Dateityp immer mit dem ausgewählten Programm geöffnet wird. Wenn Sie anschließend den Dialog abschicken, wird die Dateierdung `.java` registriert und mit Notepad als Standardverarbeitungsprogramm verknüpft. Danach können Sie `.java`-Dateien per Doppelklick in Notepad laden und werden nie wieder Ärger mit an Java-Dateien angehängten `.txt`-Dateierdungen haben.

7. Öffnen Sie ein Konsolenfenster.

Im Gegensatz zu Linux-Anwendern sind viele Windows-Anwender heutzutage gar nicht mehr mit dem Umgang mit der Konsole vertraut. Unter Windows heißt die Konsole meist „Eingabeaufforderung“ und wird am einfachsten durch die Suche nach *Eingabeaufforderung* aufgerufen. (Alternativ finden Windows-10-Anwender die Eingabeaufforderung im **Alt+X**-Menü bzw. in der App-Liste unter der Kategorie **Windows-System** und Windows-7-Anwender im **Start/Alle Programme/Zubehör**-Menü.)

Unter Linux heißt die Konsole oft auch „**Terminal**“ und kann über entsprechende Desktop-Symbole oder Links im Start-Menü geöffnet werden.

Die Konsole ist ein zeilenorientiertes Befehlseingabefenster, in deren jeweils letzten Zeile Sie einen Systembefehl eingeben und mit der **Enter**-Taste abschicken können. Beachten Sie auch den „Prompt“ zu Beginn der Zeile.

Der Prompt zeigt Ihnen an, dass die Konsole auf die **Eingabe** eines Befehls wartet. Das Aussehen des Prompts ist veränderbar. Meist ist er so konfiguriert, dass der Pfad zu dem aktuellen Verzeichnis und ein abschließendes `>` angezeigt werden. Hinter dem `>` tippen Sie Ihren Befehl ein.

```
C:\Java\Kap02> IhrBefehl
```



Eine kurze Einführung in die Bedienung der Konsole finden Sie als Tutorial auf unserer Website www.carpelibrum.de.

8. In der Konsole wechseln Sie mithilfe des `cd`-Befehls („change directory“) in das Verzeichnis, in dem die zu kompilierende Quelldatei steht. Nehmen wir an, es war dies das Ver-

zeichnis `C:\Java\Kap02`. Dann tippen Sie hinter dem Prompt der Konsole den Befehl `cd C:\Java\Kap024` ein und schicken ihn durch Drücken der **Enter**-Taste ab.

Tabelle 1.1 cd-Befehle der Konsole

cd-Befehl	Beschreibung
<code>E:</code>	Wechsel zum Laufwerk <code>E:</code>
<code>cd ..</code>	Wechsel in das übergeordnete Verzeichnis
<code>cd Verzeichnis</code>	Wechsel in das angegebene untergeordnete Verzeichnis
<code>cd C:/Projekte/Verzeichnis</code>	Wechsel in das angegebene Verzeichnis

Ihr Desktop sollte nun ungefähr wie in Bild 1.3 aussehen.

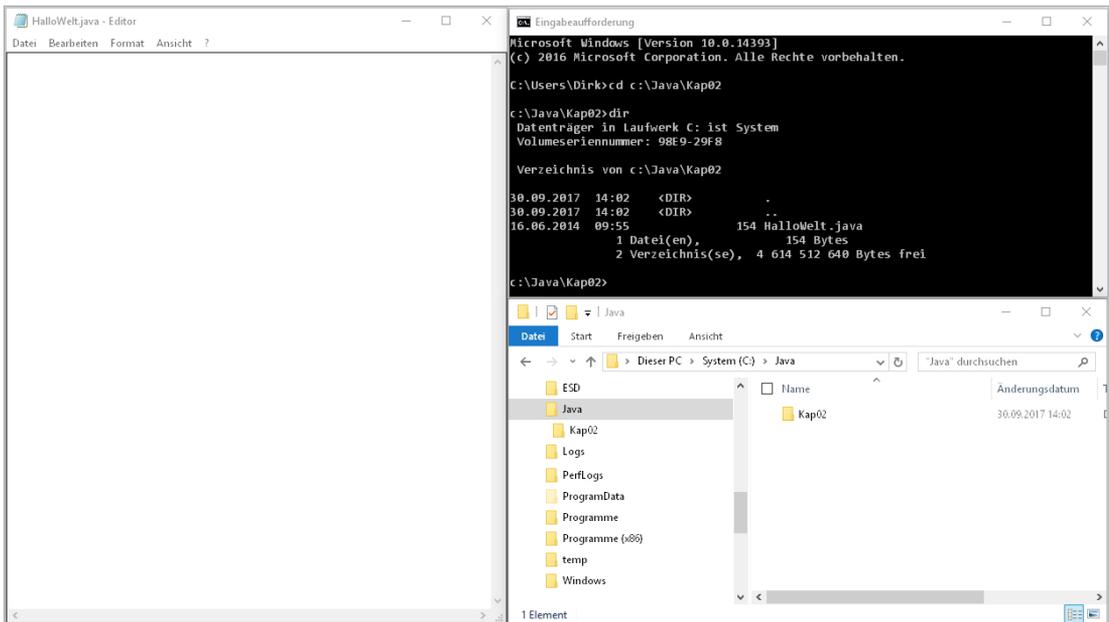


Bild 1.3 Desktop mit Editor, Konsole und Explorer (unter Windows 7)

Im Editor setzen Sie Ihren Quelltext auf. Den fertig bearbeiteten Quelltext speichern Sie im zugehörigen Programmverzeichnis.

Nach dem Speichern wechseln Sie in die Konsole, von wo aus Sie das Programm kompilieren und ausführen (die entsprechenden Befehle werden Sie gleich im nächsten Kapitel kennenlernen).

⁴ Achtung: Falls Sie die Datei auf einer anderen Partition der Festplatte abgespeichert haben als die Windows-Standardpartition `C:` (z. B. `D:`), dann müssen Sie vor dem `cd`-Befehl erst die richtige Partition anwählen, z. B. durch `d:` Enter und dann erst `cd D:\Java\Kap02!`



Nutzen Sie die Konsolen-History!

Treten bei der Kompilierung Fehler auf, korrigieren Sie die entsprechenden Zeilen im Quelltext, speichern, wechseln zur Konsole und kompilieren erneut. Die Konsolenbefehle müssen Sie dazu nicht noch einmal neu eingeben. Die Konsole merkt sich alle bereits abgeschickten Befehle in ihrer History, die Sie mithilfe der Pfeiltasten durchgehen können.

Damit sind Sie gerüstet und das Abenteuer kann beginnen!

Integrierte Entwicklungsumgebungen

Die Arbeit mit dem JDK ist wegen der vielen, nur von der Konsole aus zu bedienenden Programme etwas unhandlich. Viele Programmierer greifen daher zu integrierten Entwicklungsumgebungen, deren grafische Benutzeroberfläche den Komfort eines übergeordneten Bedienpults bietet, von dem aus sämtliche Programmierarbeiten bequem erledigt werden können. Die meisten integrierten Entwicklungsumgebungen warten daher nicht nur mit Menübefehlen zum Kompilieren und Ausführen der Programme auf, sondern auch mit

- integrierten Editoren, die speziell für die Erstellung von Quelltexten ausgelegt sind und den Programmierer mit Optionen wie Zeilennummerierung, Syntaxhervorhebung, automatischer Zeileneinzug, Codevervollständigung und Einblendung von Hilfetexten erfreuen,
- integriertem Debugger zum Aufspüren von Laufzeitfehlern,
- einer ausgeklügelten Projektverwaltung, die dem Programmierer dabei hilft, die Dateien seiner Programme (Quelltextdateien, kompilierte `.class`-Dateien, Ressourcen wie Bild- oder Sounddateien) übersichtlich zu verwalten,
- vorgefertigten Programmgerüsten und
- visueller Programmierung. (Betrifft vor allem die Erzeugung von Anwendungen mit grafischer Benutzeroberfläche (GUI). Die Fenster der Benutzeroberfläche können in einem grafischen Editor aufgebaut und bearbeitet werden, den zugehörigen Java-Code erzeugt die Entwicklungsumgebung automatisch.)

Grundsätzlich möchten wir Ihnen aber raten, anfangs auf den Einsatz einer integrierten Entwicklungsumgebung zu verzichten und rein mit dem JDK zu arbeiten. Gerade weil dem JDK der Komfort einer integrierten Entwicklungsumgebung und die Leichtigkeit und trügerische Sicherheit der visuellen Programmierung fehlt, halten wir ihn für den Einstieg ideal. Wir möchten, dass Sie erst einmal selbst lernen, wie man gute und sichere Programme schreibt, und sich nicht gleich von Anfang an auf eine Entwicklungsumgebung verlassen, die Ihnen zwar vieles abnimmt, aber auch vieles vor Ihnen verbirgt. Wir wollen nicht, dass Sie einfach aufs Geratewohl programmieren, Sie sollten auch verstanden haben, was Sie programmieren. Später können Sie dann jederzeit auf eine integrierte Entwicklungsumgebung umsteigen.

2

Der erste Kontakt

Bestimmt sind Sie schon ganz gespannt und wollen nun endlich wissen, wie man mit Java Programme schreiben kann. Wir werden uns daher auch nicht mehr mit langen Vorreden aufhalten, sondern gleich mit einem einfachen Beispiel beginnen – einer Java-Anwendung, die Sie mit einem freundlichen Hallo von der Konsole aus begrüßt. Das Programm wird nur aus wenigen Zeilen Quelltext bestehen, aber diese werden es in sich haben. Sie sollten darüber jedoch nicht erschrecken und Sie brauchen sich auch nicht darum zu sorgen, wie es mit der Java-Programmierung weitergeht, wenn schon die einfachsten Programme so kompliziert und unverständlich sind. Das Problem, speziell für uns als Autoren, liegt darin, dass man in Java auch für die einfachsten Programme auf eine Reihe weit fortgeschrittener Konzepte vorgreifen muss. Versuchen wir einfach, aus der Not eine Tugend zu machen. Anstatt gleich alles bis ins Detail verstehen zu wollen, verschaffen wir uns erst einmal einen Überblick.

■ 2.1 Die erste Java-Anwendung

In vielen Lehrbüchern über Programmiersprachen beginnt man mit dem Erstellen eines kleinen Programms, das die Meldung „Hello World“ auf den Bildschirm ausgibt. Wir wollen uns dieser Tradition anschließen und eine Anwendung erzeugen, die sich mit einem freudigen „Hallo Welt“ meldet.

Und so sieht das Programm aus:

Listing 2.1 HalloWelt

```
// Dies ist die erste Anwendung

public class HalloWelt {
    public static void main (String[] args) {
        System.out.println("Hallo Welt!");
    }
}
```

Anwendungen erstellen und ausführen

Um diese – und jede andere – Anwendung auf Ihrem Computer zu erstellen und auszuführen, gehen Sie folgendermaßen vor:

1. Öffnen Sie Ihren Texteditor.

Rufen Sie einen beliebigen Texteditor auf, mit dem Sie Ihren Quellcode als reinen Text (ASCII oder ANSI, kein Unicode) abspeichern können.

2. Geben Sie den Java-Quelltext ein.

Legen Sie in Ihrem Editor eine neue Datei an, tippen Sie obigen Quelltext ein und speichern Sie die Datei unter dem Namen *HalloWelt.java*.

Wichtig ist dabei, dass die Quelltextdatei exakt den gleichen Namen trägt wie die in dem Quelltext definierte `public`-Klasse (hier also `HalloWelt`), wobei auch die Groß- und Kleinschreibung zu beachten ist.

Weiterhin wichtig ist, dass die Datei die Dateiergung *.java* trägt und der Editor nicht eigenmächtig eine eigene Dateiergung anhängt (vgl. Anmerkung zu Notepad in Schritt 5 von Abschnitt 1.6 „Einrichtung einer eigenen Entwicklungsumgebung“).



Achtung!

Zu Anfang sollten Sie die Quelltexte bitte nicht aus der Beispielsammlung (siehe Anhang F) kopieren, sondern die Texte wirklich selbst eintippen. Es werden sich dabei zwar manche Tippfehler einschleichen, doch aus der Beseitigung dieser Fehler lernen Sie! Greifen Sie auf die Quelltexte der Beispielsammlung nur als letzte Referenz zurück, wenn Sie Ihre Programme gar nicht zum Laufen bringen.

3. Kompilieren Sie den Quelltext.

Falls Sie es nicht bereits bei der Lektüre von Abschnitt 1.6 „Einrichtung einer eigenen Entwicklungsumgebung“ getan haben, öffnen Sie jetzt ein Konsolenfenster und wechseln Sie in das Verzeichnis Ihrer Java-Quelltextdatei. Von dort rufen Sie den Java-Compiler `javac` auf und übergeben ihm die zu kompilierende Quelltextdatei:

Prompt:> `javac HalloWelt.java` **Enter**

Dieser Aufruf erzeugt eine ausführbare Bytecode-Datei mit dem Namen der übergebenen Quelltextdatei, allerdings mit der Endung `.class` – in unserem Beispiel also *HalloWelt.class*.

Sollten Sie beim Abschicken des Befehls eine Meldung in der Form „*Befehl oder Dateiname nicht gefunden*“ erhalten, ist Ihr System nicht so eingerichtet, dass Sie die Java-Entwicklungsprogramme aus jedem beliebigen Verzeichnis aufrufen können. Sie müssen dann dem Programmnamen `javac` den vollständigen Pfad voranstellen, der zu dem Programm führt. Wenn Sie beispielsweise Java im Verzeichnis `C:\Program Files\Java\jdk-9` installiert haben, würde der Aufruf `C:\Program Files\Java\jdk-9\bin\javac` lauten. Bequemer ist es, wenn Sie den Pfad zu den Java-Entwicklungsprogrammen in Ihren Systempfad eintragen (siehe Anhang B „Installation des JDK“).