Xpert.press

Die Reihe **Xpert.press** vermittelt Professionals in den Bereichen Softwareentwicklung, Internettechnologie und IT-Management aktuell und kompetent relevantes Fachwissen über Technologien und Produkte zur Entwicklung und Anwendung moderner Informationstechnologien.

Marco Block

JAVA Intensivkurs

In 14 Tagen lernen
Projekte erfolgreich zu realisieren

Unter Mitarbeit von
Ernesto Tapia und Felix Franke

Mit 90 Abbildungen

Springer

Marco Block

Freie Universität Berlin
Fachbereich Mathematik und Informatik
Takustraße 9
14195 Berlin
block@inf.fu-berlin.de
http://www.marco-block.de

Bibliografische Information der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.ddb.de abrufbar.

ISSN 1439-5428
ISBN 978-3-540-72271-7 Springer Berlin Heidelberg New York

Satz und Herstellung: LE-TEX Jelonek, Schmidt & Vöckler GbR, Leipzig
Umschlaggestaltung: KünkelLopka Werbeagentur, Heidelberg
Gedruckt auf säurefreiem Papier 33/3180 YL – 5 4 3 2 1 0

Für meine Katrin

Vorwort

Es existieren viele gute Bücher über die Programmiersprache Java und warum sollte es sinnvoll sein, ein weiteres Buch zu schreiben? Aus meiner Sicht gibt es zwei wichtige Gründe dafür.

Während der Arbeit in den Kursen mit Studenten aus verschiedenen Fachrichtungen habe ich versucht, in Gesprächen und Diskussionen herauszufinden, welche Probleme es beim Verständnis und beim Erlernen der „ersten Programmiersprache“ gab. Ein Programmierer, dem bereits Konzepte verschiedener Programmiersprachen bekannt sind, erkennt schnell die Zusammenhänge und interessiert sich primär für die syntaktische Ausprägung der neu zu erlernenden Programmiersprache. An der *Freien Universität Berlin* habe ich einige Kurse betreut, bei denen ein Großteil der Studenten keinerlei oder kaum Erfahrung im Umgang mit Programmiersprachen besaßen. Eine Motivation für dieses Buch ist es, die Erkenntnisse und Schlüsselmethoden, die ich im Laufe der Zeit gesammelt habe, auch anderen zugänglich zu machen.

Ich bin der Ansicht, dass gerade Java als Einstiegssprache besonders gut geeignet ist. Sie ist typsicher (die Bedeutung dessen wird in Kapitel 2 klar) und verfügt im Kern über einen relativ kleinen Sprachumfang. Schon in kürzester Zeit und mit entsprechender Motivation lassen sich die ersten Softwareprojekte in Java erfolgreich und zielsicher realisieren.

Dieses Buch hat nicht den Anspruch auf Vollständigkeit, dem werden andere eher gerecht. Es wird vielmehr auf eine Ausbildung zum Selbststudium gesetzt. Ein roter Faden, an dem sich dieses Buch orientiert, versetzt den Leser in nur 14 Tagen in die Lage, vollkommen eigenständig Programme in Java zu entwickeln. Zu den sehr vielseitigen Themengebieten gibt es viele Beispiele und Aufgaben. Ich habe darauf verzichtet, „Standardbeispiele“ zu verwenden und versucht, auf frische neue Anwendungen und Sichtweisen zu setzen.

Mitarbeiter dieses Buches

Motivation und Ausdauer, aus diesem über die Jahre zusammengestellten Manuskript, ein komplettes Buch zu erarbeiten, sind *Ernesto Tapia* und *Felix Franke* zu verdanken. Beide haben nicht nur mit ihrem Engagement bei dem Aufspüren von Fehlern und Ergänzungen ihren Teil beigetragen, sondern jeweils ein Kapitel beigesteuert und damit das Buch abwechslungsreicher gestaltet.

Ernesto Tapia, der auf dem Gebiet der Künstlichen Intelligenz promoviert hat, entwarf das Tetris-Projekt für Kapitel 14 „Entwicklung einer größeren Anwendung“ und hatte großen Einfluss auf Kapitel 13 „Methoden der Künstlichen Intelligenz“. Für die seit vielen Jahren in Forschung und Lehre währende gemeinsame Arbeit und Freundschaft bin ich ihm sehr dankbar.

Felix Franke war vor Jahren selbst einer meiner Studenten, dem ich den Umgang mit Java vermittelte. Durch seine Zielstrebigkeit konnte er sein Studium vorzeitig beenden und promoviert jetzt auf dem Gebiet „Neuronale Informationsverarbeitung“. Er hat neben dem Kapitel 12 „Bildverarbeitung“ viele neue Ideen beigesteuert.

Zusatzmaterialien

Neben den Beispielprogrammen und Aufgaben aus diesem Buch steht eine ständig wachsende Sammlung an kommentierten Programmen und weiteren Aufgaben zum Download zur Verfügung.

http://www.marco-block.de

Es gibt auch ein Forum, in dem Fragen erörtert und Informationen ausgetauscht werden können.

Übersicht der Kapitel

Da die Größe der einzelnen Kapitel etwas variiert, sei dem Leser angeraten auch mal zwei kleine Kapitel an einem Tag durchzuarbeiten, wenn der Stoff keine Schwierigkeiten bereitet. An kniffligen Stellen, wie z. B. der Einführung in die Objektorientierung in Kapitel 7, kann dann mehr Zeit investiert werden. Die Erfahrung zeigt, dass der Lehrstoff dieses Buches in 14 Tagen sicher aufgenommen und erfasst werden kann.

Kapitel 1 soll die Motivation zum Selbststudium wecken und unterstützt die Bereitstellung und Inbetriebnahme einer funktionsfähigen Java-Umgebung.

Kapitel 2 führt behutsam in die grundlegenden Prinzipien der Programmentwicklung ein. Die kleinsten Java-Bausteine werden vorgestellt und es wird damit das Fundament für das Verständnis der Programmierung gelegt.

In Kapitel 3 werden die zuvor eingeführten Prinzipien in Java anhand konkreter Beispiele umgesetzt. Für die praktische Arbeit wird die Klasse zunächst nur als Programmrumpf interpretiert.

Kapitel 4 behandelt das Ein- und Auslesen von Daten. Diese Daten können in Dateien vorliegen oder dem Programm auf der Konsole übergeben werden.

In Kapitel 5 wird der Umgang mit Arrays und Matrizen durch das erste Projekt ***Conway's Game of Life*** vermittelt.

Bevor mit der Objektorientierung begonnen wird, zeigt Kapitel 6 auf, welche Regeln bei der Erstellung von Programmen zu beachten sind und mit welchen Hilfsmitteln Fehler gefunden werden können.

In Kapitel 7 wird das Klassenkonzept vorgestellt. Mit Hilfe eines ***Fußballmanagers*** wird das Konzept der Vererbung vermittelt. Zusätzlich werden bisher ausgeklammerte Fragen aus den vorhergehenden Kapiteln zum Thema Objektorientierung an dieser Stelle aufgearbeitet.

Java verfügt im Kern über einen relativ kleinen Sprachumfang. Die Sprache lässt sich durch Bibliotheken beliebig erweitern. Kapitel 8 zeigt die Verwendung von Bibliotheken. Ein ***Lottoprogramm*** und das Projekt ***BlackJack*** werden mit den bisher kennengelernten Hilfsmitteln realisiert.

Kapitel 9 gibt Schritt für Schritt eine Einführung in die Erstellung von grafischen Oberflächen und die Behandlung von Fenster- und Mausereignissen.

In Kapitel 10 wird neben einer Kurzeinführung in HTML das Konzept von Applets vermittelt. Es genügen oft nur einfache Änderungen, um aus einer Applikation ein Applet zu machen.

Einen Einstieg in die Techniken der Programmentwicklung gibt Kapitel 11. Es werden viele verschiedene Programmbeispiele zu den jeweiligen Entwurfstechniken vorgestellt.

Kapitel 12 macht einen Ausflug in die Bildverarbeitung. ***Fraktale*** werden gezeichnet und verschiedene Techniken der Bildverarbeitung aufgezeigt.

Kapitel 13 beschäftigt sich mit Aspekten der Künstlichen Intelligenz, wie z. B. der ***Erkennung handgeschriebener Ziffern*** oder der Funktionsweise eines perfekt spielenden ***TicTacToe***-Spiels.

Abschließend werden in Kapitel 14 alle Phasen einer Projektentwicklung für eine Variante des Spiels ***Tetris***, vom Entwurf über die Implementierung bis hin zur Dokumentation, dargestellt.

Um den Leser für neue Projekte zu motivieren, werden in Kapitel 15 weitere Konzepte der Softwareentwicklung kurz erläutert.

Danksagungen

Mein Dank gilt allen, die auf die eine oder andere Weise zur Entstehung dieses Buches beigetragen haben. Dazu zählen nicht nur die Studenten, die sichtlich motiviert waren und mir Freude beim Vermitteln des Lehrstoffs bereiteten, sondern auch diejenigen Studenten, die im Umgang mit Java große Schwierigkeiten hatten. Dadurch wurde ich angeregt auch unkonventionelle Wege auszuprobieren.

Die Vorlesungen an der Freien Universität Berlin von *Klaus Kriegel*, *Frank Hoffmann* und *Raúl Rojas* haben mein Verständnis für didaktische Methodik dabei am intensivsten geprägt.

An dieser Stelle möchte ich meinen Eltern, Großeltern und meinem Bruder danken, die mich immer bei allen meinen Projekten bedingungslos unterstützen.

Besonders möchte ich mich auch bei den vielen kritischen Lesern, die zahlreiche Korrekturen und Verbesserungen beigesteuert haben, und für die vielen bereichernden Diskussionen bedanken (in alphabetischer Reihenfolge): *Maro Bader*, *Anne*, *Katrin*, *Inge und Detlef Berlitz*, *Erik Cuevas*, *Christian Ehrlich*, *Dominic Freyberg*, *Niklaas Görsch*, *Christine Gräfe*, *Ketill Gunnarson*, *Tobias Hannasky*, *Julia und Thorsten Hanssen*, *Frank Hoffmann*, *Nima Keshvari*, *André Knuth*, *Raul Kompaß*, *Falko Krause*, *Jan Kretzschmar*, *Klaus Kriegel*, *Tobias Losch*, *Günter Pehl*, *Marte Ramírez*, *Raúl Rojas*, *Michael Schreiber*, *Bettina Selig*, *Sonja und Thilo Rörig*, *Alexander Seibert*, *Manuel Siebeneicher*, *Mark Simon*, *Ole Schulz-Trieglaff*, *Tilman Walther*, *Daniel Werner* und *Daniel Zaldivar*.

Unterstützung erhielt ich von *Sonja Rörig*, die neben der Arbeit als Illustratorin und Designerin, Zeit für meine Abbildungen fand. Das Vererbungsbeispiel mit den witzigen Fußballspielern hat es mir besonders angetan. Vielen lieben Dank für Deine professionelle Hilfe! Auch *Tobias Losch* ist in diesem Zusammenhang noch einmal zu nennen, denn neben den zahlreichen Korrekturhilfen, hat auch er bei einigen Abbildungen geholfen und die Webseite zum Buch entworfen.

Die Zusammenarbeit mit dem Springer-Verlag, gerade bei diesem ersten Buchprojekt, war sehr angenehm und bereichernd. Für die geduldige und fachkundige Betreuung von *Hermann Engesser* und *Gabi Fischer* bin ich sehr dankbar.

Ich wünsche dem Leser genauso viel Spaß beim Lesen wie ich es beim Schreiben hatte und hoffe auf offene Kritik und weitere Anregungen.

Marco Block *Berlin, im Juli 2007*

Inhaltsverzeichnis

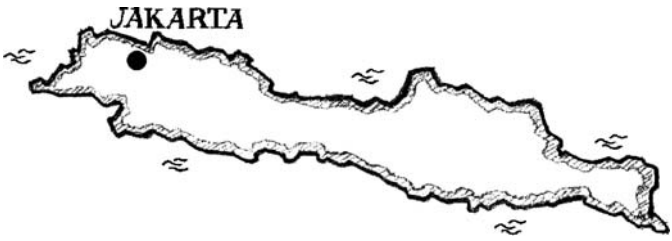
1

Tag 1: Vorbereitungen und Motivation

Unser erster Abschnitt beschäftigt sich mit der Installation von Java, der Wahl einer Entwicklungsumgebung und der Motivation zum Selbststudium. Ein innerer Antrieb und viele praktische Übungen sind unerlässlich für das Erlernen einer Programmiersprache. Programmieren heißt auch meistens, mit anderen oder für andere zu programmieren. Daher werden wir lernen, durch einen verständlichen Programmaufbau, mit Diagrammen und ausreichender Kommentierung, die Kommunikation mit anderen zu verbessern, um allein oder gemeinsam komplexe Projekte meistern zu können.

1.1 Motivation: Warum gerade Java?

Es gibt viele Gründe mit dem Programmieren zu beginnen, aber warum sollte es gerade die Programmiersprache Java sein?



Java ist auch eine indonesische Insel mit der Hauptstadt Jakarta.

Die Vorteile von Java liegen in dem vergleichsweise zu anderen Programmiersprachen relativ kleinen Befehlsumfang und der strikten Typsicherheit. Java ist plattformunabhängig und kann daher auf verschiedenen Betriebssystemen eingesetzt werden. Es gibt eine sehr große Java-Community und die Sprache ist leicht zu erlernen.

Generell gilt: Die Chance in Java etwas falsch zu machen ist sehr viel kleiner als z. B. bei C++.

Es gibt aber auch Nachteile. Geschwindigkeitsrelevante Softwaresysteme sollten nicht mit Java, sondern eher mit Programmiersprachen wie C oder C++ geschrieben werden[1]. Der Speicher kann nicht direkt angesprochen werden, alles wird über eine virtuelle Maschine gesteuert. Damit ist auch sichergestellt, dass sicherheitsproblematische Speicherabschnitte nicht einfach so ausgelesen werden können (siehe dazu Abschnitt 1.3.9 in [37]).

1.1.1 Programme für Webseiten

Neben den üblichen **Applikationen**, das sind selbstständige Anwendungsprogramme (stand-alone Applications), können wir mit Java aber auch Programme schreiben, die in eine Webseite eingebunden werden können. Diese Programme tragen den Namen **Applets**.

In diesem Buch werden wir beide Programmtypen kennenlernen und mit ihnen arbeiten. Wir werden sehen, dass oft nur wenige Arbeitsschritte notwendig sind, um eine Applikation in ein Applet zu ändern.

1.2 Vorteile des Selbststudiums

Es gibt zwei Ansätze beim Erlernen einer Programmiersprache. Der erste ist ein detailliertes Studium aller Funktionen und Eigenschaften einer Sprache und das fleißige Erlernen aller Vokabeln. Dabei können zu jedem Teilaspekt Beispiele angebracht und Programmierübungen gelöst werden. Diese Strategie könnte man als ***Bottom-Up-Lernen*** bezeichnen, da erst die kleinen Bausteine gelernt und später zu großen Projekten zusammengefügt werden.

Beim zweiten Ansatz werden die zu erlernenden Bausteine auf ein Minimum beschränkt. Es werden eher allgemeine Konzepte als speziell in einer Sprache existierende Befehle und Funktionen vermittelt. Mit diesem Basiswissen und verschiedenen Wissensquellen, die zu Rate gezogen werden können, lassen sich nun ebenfalls erfolgreich größere Projekte realisieren. Die entsprechenden Bausteine werden während des Entwicklungsprozesses studiert und erarbeitet. Das zweite Verfahren, wir könnten es ***Top-Down-Lernen*** nennen, eignet sich für Personen, die sich nicht auf eine spezielle Programmiersprache beschränken und schnell mit dem Programmieren beginnen wollen. Der Lernprozess findet dabei durch die praktische Handhabung der Sprache statt.

Eine Analogie lässt sich zum Erlernen von Fremdsprachen ziehen. Man kann beginnen, indem man alle Vokabeln paukt und dann versucht, in entsprechenden Situationen die Sätze zusammenzubauen. Der Lernprozess findet also primär vor der

[1] Trotzdem hat sich Java gerade in der Mobilkommunikation durchgesetzt.

praktischen Ausübung statt. Der zweite Weg führt erst zu einer Situation und liefert dann die entsprechenden Vokabeln, die es nachzuschlagen gilt. Man kann früher mit dem Sprechen anfangen, da sich der Lernprozess in die praktische Ausübung verlagert hat.

Ich bevorzuge die Top-Down-Variante und habe mit Bedacht diesen didaktischen Weg für die Konzeption dieses Buches gewählt. Aus diesem Grund wird der Leser schnell mit dem Programmieren beginnen können, ohne erst viele Seiten Theorie studieren zu müssen. Da dieses Buch nicht den Anspruch auf Vollständigkeit hat, wird dem Leser im Laufe der ersten Tage vermittelt, wie und wo Informationsquellen zu finden sind und wie man sie verwendet, um Probleme zu lösen.

Es ist unmöglich, ein Buch zu schreiben, das auf alle erdenklichen Bedürfnisse eingeht und keine Fragen offen lässt. Deshalb ist es ratsam, zu entsprechenden Themenbereichen zusätzliche Quellen zu Rate zu ziehen. An vielen Stellen wird darauf verwiesen. Dieses Buch eignet sich nicht nur als Einstieg in die Programmierung mit Java, sondern versucht auch einen Blick auf den ganzen Prozess einer Softwareentwicklung zu geben.

Java ist nur ein Werkzeug zur Formulierung von Programmen. Zur erfolgreichen Realisierung von Projekten gehört aber noch viel mehr. Projekte müssen genau geplant und vor der eigentlichen Erstellung möglichst so formuliert werden, dass jeder Softwareentwickler weiß, was er zu tun hat.

Dieses Buch soll dem Leser einen umfassenden Einstieg in den gesamten Prozess der Softwareentwicklung ermöglichen.

1.3 Installation von Java

Wir beginnen zunächst damit, uns mit dem System vertraut zu machen. Dazu installieren wir auf dem vorhandenen Betriebssystem eine aktuelle Java-Version[2]. Es ist darauf zu achten, dass diese eine SDK-Version ist (SDK=Software Development Kit oder JDK=Java Development Kit). Zu finden ist sie zum Beispiel auf der Internetseite von Sun Microsystems [50]:

http://java.sun.com/

Eine wichtige Anmerkung an dieser Stelle: Bei vielen Betriebssystemen ist es notwendig, die **Umgebungsvariablen** richtig zu setzen. Bei Windows XP beispielsweise geht man dazu in die „Systemsteuerung“ und dort auf „Systemeigenschaften“, dann müssen unter der Rubrik „Erweitert“ die Umgebungsvariablen geändert werden. Es gibt eine Variable *PATH*, die um den Installationsordner + „`/bin`“ von Java

[2] Die Programme dieses Buches wurden mit Java 1.5 erstellt. Es wurde versucht, allgemein gültige Funktionen und Konzepte zu verwenden.

erweitert werden muss. In älteren Systemen muss noch eine neue Umgebungsvariable mit dem Namen *CLASSPATH* erzeugt und ihr ebenfalls der Javapfad (zusätzlich zum Javapfad muss „;.;“ angehängt werden) zugewiesen werden.

Nach der Installation von Java müssen wir uns für eine Entwicklungsumgebung entscheiden und diese ebenfalls installieren. Es gibt eine Reihe von Umgebungen, die es dem Programmierer vereinfachen, Programme in Java zu entwickeln. Hier sind ein paar kostenfrei erhältliche Programme aufgelistet:

- ✔ Jeder normale Texteditor kann verwendet werden
- ✔ Eclipse (online [52])
- ✔ JCreator (online [53])
- ✔ NetBeans (online [54])
- ✔ Borland JBuilder (online [55])

Es gibt viele weitere Editoren für Java. Da ich keinen bevorzugen, sondern den angehenden Programmierer für die internen Prozesse sensibilisieren möchte, verzichte ich auf den Einsatz professioneller Entwicklungsumgebungen und schreibe alle Programme mit einem einfachen Texteditor. Dem Leser sei es selbst überlassen, eine Wahl zu treffen. Eine Übersicht findet sich beispielsweise auf der Internetseite vom Java-Tutor [51].

1.4 Testen wir das Java-System

Bevor es mit der Programmierung losgeht, wollen wir das System auf Herz und Nieren prüfen. Dazu starten wir eine **Konsole** (unter Windows heißt dieses Programm **Eingabeaufforderung**). Wenn die Installation von Java vollständig abgeschlossen ist und die Umgebungsvariablen richtig gesetzt sind, dann müsste die Eingabe der Anweisung *javac* zu folgender (hier gekürzter) Ausgabe führen:

```
C:\>javac
Usage: javac <options> <source files>
where possible options include:
  -g                         Generate all debugging info
  -g:none                    Generate no debugging info
  -g:{lines,vars,source}     Generate only some debugging info
  -nowarn                    Generate no warnings
  -verbose                   Output messages about what the compiler ...
  -deprecation               Output source locations where deprecate ...
  ...
```

So oder so ähnlich sollte die Ausgabe aussehen. Falls eine Fehlermeldung an dieser Stelle auftaucht, z. B.

```
C:\>javac
Der Befehl "javac" ist entweder falsch geschrieben oder
konnte nicht gefunden werden.
```

bedeutet dies, dass das Programm *javac* nicht gefunden wurde. An dieser Stelle sollte vielleicht die *PATH*-Variable noch einmal überprüft (siehe dazu Abschnitt 1.3) oder ein Rechnerneustart vorgenommen werden.

Testen wir unser System weiter. Um herauszufinden, ob der *CLASSPATH* richtig gesetzt ist, schreiben wir ein kurzes Programm. Dazu öffnen wir einen Editor und schreiben folgende Zeile hinein:

```
1 public class Test{}
```

Diese Datei speichern wir mit dem Namen **Test.java** in einem beliebigen Ordner. Am besten wäre an dieser Stelle vielleicht, die Datei einfach auf „`C:\`“ zu speichern, denn wir werden das Programm anschließend sowieso wieder löschen. In meinem Fall hatte ich einen Ordner mit dem Namen „`Java`“ in „`C:\`“ angelegt und dort hinein die Datei gespeichert.

Anschließend navigieren wir in der Konsole zu der Datei und geben die folgende Anweisung ein:

```
C:\>cd Java
C:\Java>javac Test.java

C:\Java>
```

Wenn keine Fehlermeldung erscheint, ist das System bereit und wir können endlich mit der Programmierung beginnen.

Dieses kleine Beispiel hat uns einen ersten Einblick in die Programmerstellung mit Java gegeben. Wir werden in einem Editor die Programme schreiben und sie anschließend in der Konsole starten. Mit der Anweisung *javac* wird das Programm in den sogenannten Byte-Code umgewandelt, dabei wird eine Datei mit dem gleichen Namen erzeugt, aber mit der Endung „`.class`“ versehen. Eine solche Datei wird auch für unser kleines Beispiel erzeugt.

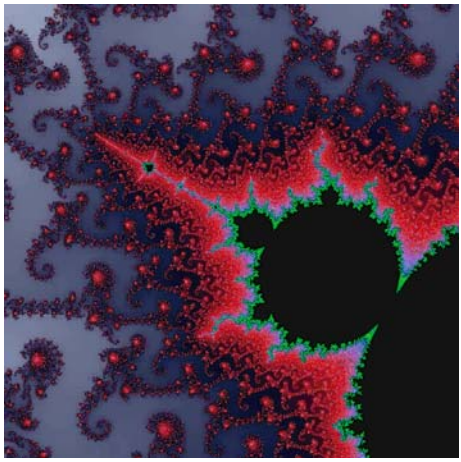
```
C:\Java>javac Test.java

C:\Java>dir
 Volume in Laufwerk C: hat keine Bezeichnung.
 Volumeseriennummer: A8DB-F3AF

 Verzeichnis von C:\Java

14.03.2007  12:22    <DIR>          .
14.03.2007  12:22    <DIR>          ..
14.03.2007  12:22               182 Test.class
14.03.2007  12:10                19 Test.java
               2 Datei(en)            201 Bytes
               2 Verzeichnis(se), 372.468.928.512 Bytes frei
```

Später werden wir noch erfahren, wie wir diesen Byte-Code starten und beispielsweise diese schicke Ausgabe erzeugen können:



Ich hoffe, dass Motivation und Kreativität für die Realisierung von Projekten in Java jetzt ausreichend vorhanden sind. Wir werden im folgenden Kapitel mit den kleinsten Bausteinen beginnen.

2

Tag 2: Grundlegende Prinzipien der Programmentwicklung

Um einen Einstieg in die Programmierung zu wagen, müssen wir uns zunächst mit den primitiven Datentypen und den Prinzipien der Programmentwicklung vertraut machen. In Java wird im Gegensatz zu anderen Programmiersprachen, wie z. B. C oder C++, besonderer Wert auf Typsicherheit gelegt. Damit ist gemeint, dass das Programm bei der Verwaltung von Speicher immer genau Bescheid wissen möchte, um welchen Datentyp es sich handelt. Im Anschluss daran können wir mit den ersten Programmierübungen beginnen.

2.1 Primitive Datentypen und ihre Wertebereiche

Mit Programmen wollen wir den Computer dazu bringen, bestimmte Aufgaben und Probleme für uns zu lösen. Wir haben es dabei immer mit Daten zu tun, die entweder als Ein- oder Ausgabe für die Lösung einer Aufgabe oder zum Speichern von Zwischenlösungen verwendet werden.

Die vielen Jahre der Softwareentwicklung haben gezeigt, dass es drei wichtige Kategorien von Daten gibt, die wir als atomar bezeichnen können, da sie die kleinsten Bausteine eines Programms darstellen und wir sie später zu größeren Elementen zusammenfassen werden.

Wahrheitswerte

Wahrheitswerte repräsentieren die kleinste Einheit im Rechner. Auf 0 oder 1, *wahr* oder *falsch*, Strom *an* oder *aus* beruhen alle technischen Ereignisse eines Rechners. In jedem Programm werden sie verwendet, ob bewusst oder unbewusst.

Zeichen, Symbole

Die Ein- und Ausgaben von Text sind ebenfalls wichtige Hilfsmittel für einen Programmierer. Zeichen oder Symbole lassen sich später zu einem größeren Datentyp (Zeichenkette) zusammenfassen.

Zahlen

Zahlen sind wichtig, um beispielsweise Ereignisse zu zählen. Je nach Verwendungszweck beanspruchen Zahlen mal mehr und mal weniger Speicher im Rechner. Eine Gleitkommazahl, die z. B. einen Börsenkurs wiedergibt, benötigt mehr Speicher als eine ganze Zahl, die die Stunden einer Uhr repräsentiert.

2.1.1 Primitive Datentypen in Java

Je nach Programmiersprache werden nun verschiedene Datentypen aus diesen drei Kategorien angeboten. Wir nennen sie, da sie die kleinsten Bausteine sind, primitive Datentypen. In Java gibt es:

Wahrheitswerte: `boolean`

Zeichen, Symbole: `char`

Zahlen: `byte`, `short`, `int`, `long`, `float`, `double`

Wie diese Übersicht zeigt, wird in der Programmierung dem Bereich der Zahlen eine besondere Aufmerksamkeit gewidmet. Das hat gute Gründe. Um gute, schnelle Programme zu schreiben, ist es notwendig, sich Gedanken über die verwendeten Datentypen zu machen. Sicherlich könnte der Leser der Meinung sein, dass ein Zahlentyp ausreicht, beispielsweise ein `double`, der sowohl positive und negative als auch ganze und gebrochene Zahlen darzustellen vermag. Das ist korrekt.

Aber ein `double` benötigt im Rechner jede Menge Speicher und für Operationen, wie Addition, Subtraktion, Multiplikation und Division, einen größeren Zeitaufwand als z. B. ein `short`. Da wir in Programmen meistens sehr viele Datentypen und Operationen auf diesen verwenden wollen, um Aufgaben zu lösen, gilt die Devise, immer den Datentyp zu verwenden, der für die Aufgabe geeignet ist und den kleinsten Speicherbedarf hat.

Um nun entscheiden zu können, welche Datentypen in Frage kommen, bleibt es uns nicht erspart einmal einen Blick auf die Wertebereiche zu werfen:

Datentyp	Wertebereich	BIT
boolean	true, false	8
char	0 ... 65.535 (z. B. 'a','b',...,'A',...'1',...)	16
byte	-128 bis 127	8
short	-32.768 bis 32.767	16
int	-2.147.483.648 bis 2.147.483.647	32
long	-9.223.372.036.854.775.808 bis 9.223.372.036.854.775.807	64
float	+/-1,4E-45 bis +/-3,4E+38	32
double	+/-4,9E-324 bis +/-1,7E+308	64

Übersicht der primitiven Datentypen in Java [38]. Die dritte Spalte zeigt den benötigten Speicher in BIT.

Diese Tabelle muss man jetzt nicht auswendig lernen, es genügt zu wissen, wo man sie findet. Von einer Programmiersprache zu einer anderen, selbst bei verschiedenen Versionen einer Programmiersprache, können sich diese Werte unterscheiden, oder es kommen neue primitive Datentypen hinzu.

Dem Leser mag aufgefallen sein, dass für einen `boolean`, obwohl er nur zwei Zustände besitzt, 8 BIT reserviert sind. Logischerweise benötigt ein `boolean` nur 1 BIT zur Speicherung der zwei möglichen Zustände `true` und `false`, aber aus technischen Gründen sind 8 BIT die kleinste Speichereinheit in der aktuellen Rechnergeneration.

2.2 Variablen und Konstanten

2.2.1 Deklaration von Variablen

Um die Datentypen aus dem vorhergehenden Abschnitt verwenden zu können, müssen wir, wie es z. B. in der Algebra üblich ist, **Variablen** verwenden. Diese Variablen dienen als Platzhalter für die Werte im Speicher und belegen je nach Datentyp mehr oder weniger Ressourcen. Um eine Variable eines bestimmten Typs im Speicher anzulegen, die für den Rest des Programms (oder bis sie gelöscht wird) bestehen bleibt, müssen wir sie **deklarieren**.

```
<Datentyp> <name>;
```

Beispielsweise wollen wir einen Wahrheitswert verwenden:

```
boolean a;
```

Die Variable `a` steht nach der **Deklaration** bereit und kann einen Wert zugewiesen bekommen. Das nennen wir eine **Zuweisung**.

Wir könnten nun nacheinander Variablen deklarieren und ihnen anschließend einen Wert zuweisen. Es lassen sich auch mehrere Variablen eines Datentyps gleichzeitig deklarieren oder, wie es die letzte Zeile demonstriert, Deklaration und Zuweisung in einer Anweisung ausführen.

```
boolean a;
a = true;
int b;
b = 7;
float c, d, e;
char f = 'b';
```

Die Bezeichnung einer Variable innerhalb des Programms wird uns überlassen. Es gibt zwar ein paar Beschränkungen für die Namensgebung, aber der wichtigste Grund für die Wahl ist die Aufgabe der Variable. Ein paar Beispiele:

```
boolean istFertig;
double kursWert;
int schrittZaehler;
```

Die Beschränkungen für die Bezeichnungen in Java lauten:

Variablennamen beginnen mit einem Buchstaben, Unterstrich oder Dollarzeichen (nicht erlaubt sind dabei Zahlen!). Nach dem ersten Zeichen dürfen aber sowohl Buchstaben als auch Zahlen folgen. **Operatoren** und **Schlüsselwörter** dürfen ebenfalls nicht als Variablennamen verwendet werden.

In diesem Buch werden bei allen Programmbeispielen die Schlüsselwörter fett gekennzeichnet. Diese Form der Kennzeichnung nennt man **Syntaxhervorhebung** und sie fördert die Übersicht und Lesbarkeit von Programmen.

Reservierte Schlüsselwörter

`abstract`, `assert`, `boolean`, `break`, `byte`, `case`, `catch`, `char`, `class`, `const`, `continue`, `default`, `do`, `double`, `else`, `enum`, `extends`, `false`, `final`, `finally`, `float`, `for`, `future`, `generic`, `goto`, `if`, `implements`, `import`, `inner`, `instanceof`, `int`, `interface`, `long`, `native`, `new`, `null`, `operator`, `outer`, `package`, `private`, `protected`, `public`, `rest`, `return`, `short`, `static`, `strictfp`, `super`, `switch`, `synchronized`, `this`, `throw`, `throws`, `transient`, `true`, `try`, `var`, `void`, `volatile`, `while`

Noch ein Hinweis an dieser Stelle: Java ist eine sogenannte **textsensitive Sprache**, was bedeutet, dass auf Groß- und Kleinschreibung geachtet werden muss. Folgendes würde also nicht funktionieren:

```
boolean istFertig;
istFERTIG = true;
```

Damit wären wir also in der Lage, Programme quasi wie einen Aufsatz hintereinander weg zu schreiben. Durch eine überlegte Variablenbezeichnung werden zwei Dinge gefördert. Zum einen benötigt der Programmautor weniger Zeit, um seine eigenen Programme zu lesen, deren Erstellung vielleicht schon etwas länger zurück liegt, und zum anderen fördert es die Zusammenarbeit mit anderen Programmierern.

Im Laufe der Zeit haben sich einige Konventionen bezüglich der Erstellung von Programmen etabliert. Man sollte sich an diese halten, um sich selbst und anderen das Leben nicht unnötig schwer zu machen. Frei nach dem Motto: Wir müssen nicht immer alles machen, was erlaubt ist.

Die gängigen Konventionen werden nach und nach in den folgenden Abschnitten beschrieben.

2.2.2 Variablen und Konstanten

Variablen sind eine schöne Sache und unverzichtbar. Manchmal ist es notwendig, Variablen zu verwenden, die die Eigenschaft besitzen, während des Programmablaufs unverändert zu bleiben. Beispielsweise könnte ein Programm eine Näherung der Zahl π (pi) verwenden. Der Näherungswert soll während des gesamten Programmablaufs Bestand haben und nicht geändert werden.

```
double pi = 3.14159;
```

Damit aber nun gewährleistet ist, dass man selbst oder auch ein anderer Softwareentwickler weiß, dass dieser Platzhalter im Speicher nicht variabel und demzufolge keine Änderung erlaubt ist, schreiben wir einfach ein `final` davor, verwenden (laut Konvention) für den Namen nur Großbuchstaben und machen so aus der Variable eine **Konstante**.

```
final <Datentyp> <NAME>;
```

```
final double PI = 3.14159;
```

Nun herrscht Klarheit. Wir haben anschließend keine Erlaubnis, in einem späteren Programmabschnitt diese Konstante zu ändern. Java kümmert sich darum und wird einen Fehler melden, falls wir es dennoch versuchen.

2.3 Primitive Datentypen und ihre Operationen

Die vorherige ausführliche Einführung diente dem Zweck, im Folgenden die **Operationen** der einzelnen Datentypen besser verstehen zu können. Die Syntax (das Aufschreiben des Programms in einem vorgegebenen Format) lenkt nun nicht mehr ab und wir können uns die wesentlichen Verknüpfungsmöglichkeiten der Datentypen anschauen.

2.3.1 boolean

Bezeichnet einen Wahrheitswert und kann demzufolge `true` (wahr) oder `false` (falsch) sein.

```
boolean b;
b = false;
```

Zunächst wird `b` als `boolean` deklariert und ihm anschließend der Wert `false` zugewiesen (mit = wird ein Wert zugewiesen). Mit dem Datentyp `boolean` lassen sich alle logischen Operationen ausführen. Es gibt eine weit verbreitete Möglichkeit, auf die wir später noch genauer eingehen werden, einen `boolean` für eine Entscheidung zu verwenden:

```
if (<boolean>) <Anweisung>;
```

Wenn der `boolean` den Wert `true` hat, dann wird die nachfolgende Anweisung ausgeführt. Im anderen Fall, wenn der `boolean` den Wert `false` hat, überspringen wir die Anweisung.

Es lassen sich aber auch, wie schon erwähnt, logische Operationen mit einem `boolean` ausführen. Wir verwenden für die folgenden Abschnitte zwei `boolean` B_1und B_2 mit ihren möglichen Belegungen 1 für `true` und 0 für `false`.

Das logische UND

Nach folgender Wertetabelle wird die logische Verknüpfung **UND** angewendet:

B_1	B_2	B_1 UND B_2
0	0	0
0	1	0
1	0	0
1	1	1

Nur wenn beide Operanden `true` sind, liefert die Operation **UND** auch ein `true`. In Java schreiben wir die Operation **UND** mit dem Symbol `&&`. Schauen wir uns ein Beispiel an:

```
boolean a, b, c;
a = true;
b = false;
c = a && b;
```

Wenn wir uns die Wertetabelle anschauen, dann sehen wir für den Fall $B_1 = 1$ und $B_2 = 0$, dass B_1 UND B_2, also `c`, den Wert 0, also `false` hat.