Florian Kottmair

# JAVA FÜR EINSTEIGER

Einführung in die Programmierung mit Java ohne Vorkenntnisse

Florian Kottmair

# JAVA FÜR EINSTEIGER

Einführung in die Programmierung mit Java ohne Vorkenntnisse

Internet: www.kottmair.net
E-Mail : java@kottmair.net

Druck und Verlag: epubli GmbH, Prinzessinnenstrasse 20, 10969 Berlin
ISBN: 978-3-8442-7117-1


Titelbild: Javafarn; Vektorgrafik „Java Ferns“ von SnEptUne auf *openclipart.org*

# INHALTSVERZEICHNIS

# VORWORT

Sie haben sich also entschieden, eine neue Sprache zu erlernen. Zwar keine Fremdsprache, jedoch eine fremde Sprache mit dem Namen *Java*. Anhand dieser Sprache möchten Sie zugleich das Programmieren lernen, also das Schreiben von Anweisungen, die ein Computer versteht und die ihn dazu bringen, etwas Sinnvolles zu tun. Sie müssen also nicht nur ein paar neue Vokabeln pauken und ein wenig Grammatik üben, sondern sich auch eine neue Art zu sprechen von Grund auf aneignen. Im Moment fühlt sich dieses Vorhaben vielleicht für Sie an, wie wenn sich ein Westeuropäer vornimmt, Singhalesisch zu lernen. Doch keine Angst! Der Wortschatz Ihrer neuen Sprache ist sehr begrenzt, die Grammatik basiert auf überschaubar wenigen Grundregeln und mit ein wenig logischem Denkvermögen und Basiskenntnissen in Mathematik und Englisch liegt die Programmiersprache Java schon sehr bald wie ein Bausatz vor Ihnen, aus dem Sie sich zielsicher die benötigten Teile heraussuchen und mit spielerischer Leichtigkeit viele interessante neue Dinge daraus erschaffen können.

Ich selbst habe Java um die Jahrtausendwende herum gelernt, mit dem umfangreichen Skript meines Informatikprofessors. Nach wenigen Seiten sollte ich bereits Anwendungsoberflächen unter Windows entwickeln, obwohl ich noch schwer mit dem Konzept der Objektorientierung an sich zu kämpfen hatte. Über die Jahre bin ich in Studium und Berufsleben noch vielen Programmier-Neulingen begegnet, die am Erlernen einer Programmiersprache verzweifelt sind, weil sie komplexen Code schreiben sollten, obwohl sie die Grundlagen noch überhaupt nicht verinnerlicht hatten, da ihnen diese von ihren Professoren, Dozenten oder Lehrbüchern viel zu flüchtig vermittelt worden waren. Doch niemand kann in einer neu erlernten Sprache eine Rede halten, wenn er noch nicht einmal den Satzbau und die grundlegende Grammatik beherrscht.

Mein Ansatz und Vorhaben ist, Ihnen die Programmiersprache Java in diesem Buch Schritt für Schritt näherzubringen, Sie mit den grundlegenden Konzepten und Sprachelementen langsam und aufeinander aufbauend vertraut zu machen und viele Dinge anhand greifbarer Beispiele aus der realen Welt und vor allem durch zahlreiche kurze Codebeispiele zu veranschaulichen. Dabei werde ich

soweit möglich immer vom Einfachen zum Speziellen übergehen und Begriffe oder Sprachelemente immer dann erklären, wenn sie gerade in den Kontext passen – und jeweils auch nur so tiefgehend, wie es aktuell für das Verständnis nötig ist. Sie werden nach der Lektüre dieses Buchs keine grafischen Oberflächen gestalten und keine Webanwendungen entwickeln können, aber Sie werden (hoffentlich) die wichtigsten Grundkonzepte des Programmierens sowie die Syntax der Programmiersprache Java kennen. Zudem werden Sie mit den am häufigsten gebrauchten Java-Datentypen und Standard-Objekten vertraut und fähig sein, eigene kleine Anwendungen zu programmieren und Ihr auf einer soliden Basis stehendes Wissen im Selbststudium zu erweitern.

Eine Plattform, um Sie während und auch nach der Lektüre dieses Buchs nicht „allein zu lassen“, habe ich in Form eines Online-Forums für Java-Einsteiger geschaffen. Dort können Sie sich mit mir und anderen Lesern austauschen und sich gegenseitig beim Entwickeln Ihrer eigenen Java-Programme unterstützen. Sie finden das Forum unter der Adresse www.javaeinsteiger.de.

Um sicherzustellen, nicht denselben Fehler wie viele andere Lehrenden zu begehen (nämlich, zu viele Dinge als „selbstverständlich“ anzusehen und daher nur ungenügend zu erklären), hatte ich es mir zum Ziel gesetzt, anhand dieses Lehrbuchs meiner Frau (sie ist Juristin) das Programmieren beizubringen. Auch wenn dabei stellenweise noch persönliche „Nachhilfe“ angefordert wurde, hat es tatsächlich funktioniert und meine Frau kann heute (innerhalb der in diesem Buch gezogenen thematischen Grenzen) Java-Programme schreiben. Ihr sei an dieser Stelle auch ganz herzlich gedankt für das konstruktive Feedback, das sie mir zukommen ließ und die Zeit, die sie investiert hat, um zu erlernen und zu verstehen, womit ich nun bereits seit vielen Jahren meine Brötchen verdiene.

Florian Kottmair

Augsburg, im Oktober 2013

# EINFÜHRUNG

## DIE GESCHICHTE DER PROGRAMMIERUNG

Die Geschichte der Computer begann etwa um das Jahr 1812, als der britische Mathematikprofessor Charles Babbage (1791 - 1871) eine mechanische Vorrichtung erdachte, mit deren Hilfe Polynomfunktionen ausgewertet werden konnten. Durch die vielversprechenden Ergebnisse der Forschungen an seiner *Differenzmaschine* ermutigt, plante er die Entwicklung eines noch allgemeineren Rechenwerks, der *Analytical Engine*. Das von einer Dampfmaschine angetriebene Gerät sollte, wie zu dieser Zeit auch bei mechanischen Webstühlen üblich, mit Hilfe von Lochkarten gesteuert werden. Aufgrund verschiedener technischer Probleme und weil er die Umsetzung nicht finanzieren konnte, wurde die *Analytical Engine* nie gebaut. Trotzdem bildeten seine Erkenntnisse die Grundlage für die Entwicklung universell programmierbarer Computer. Wichtige Pionierarbeit leistete dabei die Mathematikerin Ada Lovelace (1815 - 1852), die einen schriftlichen Plan entwarf, wie man mit Babbages *Analytical Engine* Bernoulli-Zahlen berechnen konnte. Dieser Algorithmus wird heute als erstes Computerprogramm bzw. erste Software der Welt gesehen und Lovelace ist somit als erste Programmiererin der Geschichte anerkannt. Nach ihr wurde im Jahr 1983 die strukturierte Programmiersprache *Ada* benannt.

Es dauerte noch mehr als 100 Jahre, bis aus den Theorien zur *Analytical Engine* ein erster funktionsfähiger Prototyp entstand. Die 1941 durch den Deutschen Konrad Zuse (1910 - 1995) entwickelte und in Zusammenarbeit mit Helmut Schreyer in Berlin gebaute *Z3* war der erste universell programmierbare Rechner der Welt. Leider wurde die *Z3* bereits zwei Jahre später bei einem Bombenangriff zerstört. Als weiterer Meilenstein wird der 1946 in den USA von John Presper Eckert (1919 - 1995) und John William Mauchly (1907 - 1980) vorgestellte Rechner *ENIAC* (*Electronic Numerical Integrator and Computer*) gesehen, der im Unterschied zur mechanisch betriebenen *Z3* bereits vollelektronisch arbeitete.

Um solche Maschinen (sowie alle bis heute weiterentwickelten Computer, die dennoch alle noch immer nach demselben Grundprinzip funktionieren) sinnvoll verwenden zu können, muss man ihnen mitteilen, was sie denn mit ihren Rechenwerken bzw. Prozessoren anstellen sollen. Eine solche Arbeitsvorschrift

nennt man *Programm*, das Erstellen einer solchen Reihe von Anweisungen *Programmieren*. Will man einem Computer nun aber beispielsweise befehlen, er solle 1 + 1 zusammenzählen, müsste dies in einer für den Computer verständlichen Weise grob in etwa wie folgt formuliert werden:

```
Lade die Zahl 1 in den Speicherbereich 1
Lade die Zahl 1 in den Speicherbereich 2
Addiere die Werte aus den Speicherbereichen 1 und 2
```

In *Assembler*, einer Programmiersprache, in der man die Befehle sehr ähnlich eingibt, wie sie der Computer dann auch tatsächlich verarbeiten kann, würde dies etwa so aussehen:

```
LDI R1, 1
LDI R2, 1
ADD R1, R2
```

Davon würde letztendlich in der tatsächlich für den Computer verständlichen *Maschinensprache* nur noch eine Reihe von Nullen und Einsen übrig bleiben (denn andere Zustände als „Strom fließt“ (1) oder „Strom fließt nicht“ (0) kann ein mit Strom betriebenes Gerät nicht unterscheiden). Wenn das Ergebnis dann auch auf dem Bildschirm angezeigt werden soll oder der Benutzer die zu addierenden Zahlen selbst über die Tastatur eingeben will, wären noch bedeutend mehr kleine Arbeitsanweisungen nötig. Zudem muss bei der Programmierung eines Computers so nah an der Hardwareebene stets auf die Besonderheiten des verwendeten Prozessortyps eingegangen werden. Der Programmierer muss beispielsweise genau wissen, wie viele Register (Speicherstellen) der Prozessor besitzt und wie groß diese sind. Ein so erstelltes Programm würde dann unter Umständen nicht mehr ohne Änderungen auf einer anderen Hardware ausgeführt werden können. Aus diesem Grund und weil es weitaus einfacher, schneller und weniger fehleranfällig ist, statt den oben dargestellten Befehlen einfach nur 1 + 1 zu schreiben, wurden mit der Zeit verschiedene Programmiersprachen mit einer zum Teil sehr unterschiedlichen *Syntax* (was für Anweisungen gibt es; wie wird ein korrekter Befehl formuliert) entwickelt, die einen Programmierer von der Last befreien sollen, sich mit Hardwarearchitekturen und kryptischen Befehlen auseinanderzusetzen und es ermöglichen, den Programmcode so zu formulieren, dass ihn ein Mensch leicht verstehen und nachvollziehen kann.

Die erste höhere Programmiersprache war Konrad Zuses *Plankalkül* (1946), die aber bald schon von erweiterten, aufgabenspezifischen Sprachen abgelöst wurde: *Fortran* (***For**mula **Tran**slation*, 1954) für die Übersetzung von Formeln, *ALGOL* (***Algo**rithmic **L**anguage*, 1958) als Algorithmensprache und *LISP* (***Lis**t **P**rocessing*, 1959) zur Listenverarbeitung. Mit diesen Sprachen konnten Konstrukte wie *bedingte Anweisungen* („wenn x wahr ist, dann führe y aus“) oder *Schleifen* („solange x wahr ist, führe y aus“) mit einfachen, verständlichen Befehlen codiert werden. Um solche Programme wieder in eine für den Computer verständliche Form zu bringen, wandeln *Compiler* den Code in Assemblercode und von dort aus in Maschinencode um.

Der erste Compiler, er hieß schlicht *A-0*, wurde von Grace Hopper (1906 - 1992), einer Informatikerin der US-Marine, geschrieben. Hopper, die erst im Alter von 80 Jahren aus dem Dienst ausschied, war auch maßgeblich an der Entwicklung der Programmiersprache *COBOL* (***Co**mmon **B**usiness **O**riented **L**anguage*, 1960) beteiligt, was ihr den Spitznamen „Grandma COBOL“ einbrachte.

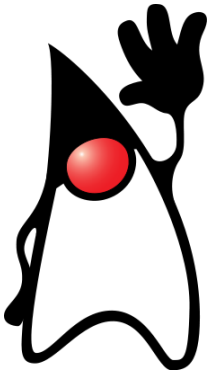
Anfang der 1970er Jahre entwickelte Dennis Ritchie (1941 - 2011), ein Informatiker der Bell Laboratories (heute AT&T), die Programmiersprache *C* (als Nachfolger der ebenfalls von ihm mitentwickelten Programmiersprache *B*) für die Programmierung des Systemkerns des Betriebssystems *Unix*. C ist eine *imperative* Programmiersprache, was bedeutet, dass der Programmablauf durch eine Abfolge einzelner Befehle vorgegeben wird, die beschreiben, was in welcher Reihenfolge erledigt werden soll. Im Unterschied dazu würde die *deklarative* Programmierung definieren, *was* ein Programm tun soll, aber nicht *wie*. C entwickelte sich schnell zu einer der am weitesten verbreiteten Programmiersprachen und beeinflusste viele später entstandene Sprachen wie beispielsweise *C++*, *C#* (sprich: „C sharp“), *PHP* oder *Java*.

## Was ist Java?

Bevor wir versuchen, ein erstes Programm in Java zu schreiben, sollen zunächst die grundlegenden Eigenschaften dieser Programmiersprache vorgestellt und näher erläutert werden.

Java wurde im Jahr 1995 von Sun Microsystems eingeführt und bezeichnet nicht nur die Programmiersprache selbst, sondern auch die dahinter stehende Technologie, mit der in Java geschriebene Anwendungen auf den unterschiedlichsten Arten von Computersystemen ausgeführt werden können. Der Name „Java“ leitet sich vom Namen eines Straßencafés („Java City – roasters of fine coffee“) her, in dem die an der Entwicklung von Java beteiligten Programmierer sich häufig ihren Kaffee holten (die beliebteste Kaffeesorte war übrigens ebenfalls *Java*).

Der Durchbruch kam mit der Integration der Java-Technologie in den damals marktführenden Internetbrowser *Netscape Navigator*, wodurch Java innerhalb kurzer Zeit einer breiten Anwenderschicht zur Verfügung stand. Im Januar 2010 übernahm die Oracle Corporation die Firma Sun Microsystems und damit auch die Rechte an Java. Heute ist Java die am zweithäufigsten verwendete Programmiersprache nach C (Quelle: *TIOBE Programming Community Index for July 2013* auf *www.tiobe.com*).



*Duke*, das Java-Maskottchen (© Sun Microsystems Inc.)

Mit der Entwicklung von Java sollten unter anderem die folgenden Ziele erreicht werden:

- **Einfachheit**: Im Vergleich zu ähnlichen Programmiersprachen wie C++ oder C# besitzt Java einen deutlich reduzierten Sprachumfang, was die Sprache zwar einerseits einfacher, aus Sicht von Kritikern jedoch gleichzeitig auch unflexibler macht.
- **Objektorientierung**: Java ist eine objektorientierte Programmiersprache, was bedeutet, dass die verwendeten, zusammengehörenden Datenstrukturen in sogenannten Objekten zusammengefasst sind. Mehr dazu im Kapitel *Klassen, Objekte und Vererbung*.
- **Vertrautheit**: Durch seine Nähe zu vergleichbaren Programmiersprachen ist Java für erfahrene Programmierer schnell erlernbar und enthält keine Konzepte oder Sprachelemente, durch die es sich radikal von diesen anderen Sprachen unterscheidet.

- **Robustheit**: Beim Design der Sprache wurden besonders fehleranfällige Sprachelemente bewusst weggelassen bzw. durch die Einführung neuer Konzepte überflüssig gemacht.
- **Sicherheit**: Java enthält einige vorgefertigte Sicherheitskonzepte, mit denen unter anderem der Zugriff auf sicherheitskritische Programmbereiche kontrolliert werden kann.
- **Architekturneutralität und Portabilität**: Java-Programme können im Prinzip auf jeder beliebigen Computerhardware ausgeführt werden (es sei denn, sie sind beispielsweise für ein spezielles Betriebssystem optimiert).
- **Parallelisierbarkeit**: Java unterstützt die Entwicklung von Anwendungen, die ihre Aufgaben gleichzeitig auf verschiedenen Recheneinheiten eines Computers ausführen lassen können (dies wird als *Multithreading* bezeichnet). Bei heutigen Systemen mit mehreren Recheneinheiten (Prozessoren oder Prozessorkernen) kann dadurch eine enorme Leistungssteigerung bzw. Zeitersparnis erreicht werden.

Im Lauf der Jahre wurde die Programmiersprache um viele neue Funktionen und Sprachelemente erweitert, mit denen neue Möglichkeiten geschaffen und das Formulieren von Programmcode vereinfacht wurde. Bis zur Version 5 gab es eine uneinheitliche Versionszählung. So wurde beispielsweise die Version Java 1.2 als „Java 2“ vermarktet, Java 1.3 und 1.4 dann wieder unter der korrekten Versionsnummer vertrieben, und seit Java 5 (interne Versionsnummer 1.5) wird nur noch die Nachkommastelle zur Benennung der Version herangezogen.

Java ist *freie Software* und steht unter der *General Public License* (*GPL*), einer weit verbreiteten Lizenz, die es jedem Benutzer erlaubt, die Software unentgeltlich zu nutzen, zu studieren, zu verbreiten und zu verändern.

# JAVA INSTALLIEREN UND EINRICHTEN

## PRÜFEN, OB JAVA BEREITS INSTALLIERT IST

Damit man mit Java arbeiten (also sowohl programmieren als auch bestehende Programme ausführen) kann, muss es zunächst installiert werden. Auf vielen Computern ist heutzutage bereits Java vorinstalliert, meistens jedoch nur eine Laufzeitumgebung (*Java Runtime*), mit der fertige Anwendungen gestartet werden können. Zum Erstellen eigener Programme benötigen wir jedoch eine Softwareentwicklungs-Version (*Software Development Kit/SDK* bzw. *Java Development Kit/JDK*) von Java, denn nur hier ist unter anderem ein *Compiler* enthalten, der die Quellprogramme in die Maschinensprache übersetzt.

Um zu überprüfen, ob Java bereits installiert ist, muss man auf die Befehlszeile wechseln. Dies erreicht man unter Windows 8, indem man die Anwendung „Eingabeaufforderung“ sucht (unter dem Symbol „Suchen“ in der Charm Bar am rechten Bildschirmrand) und startet. Unter älteren Windows-Versionen gelangt man zur Eingabeaufforderung, indem man im Startmenü den Befehl „Ausführen…“ wählt und „cmd“ eintippt.


Suchen der Anwendung „Eingabeaufforderung“ über die Charm Bar von Windows 8

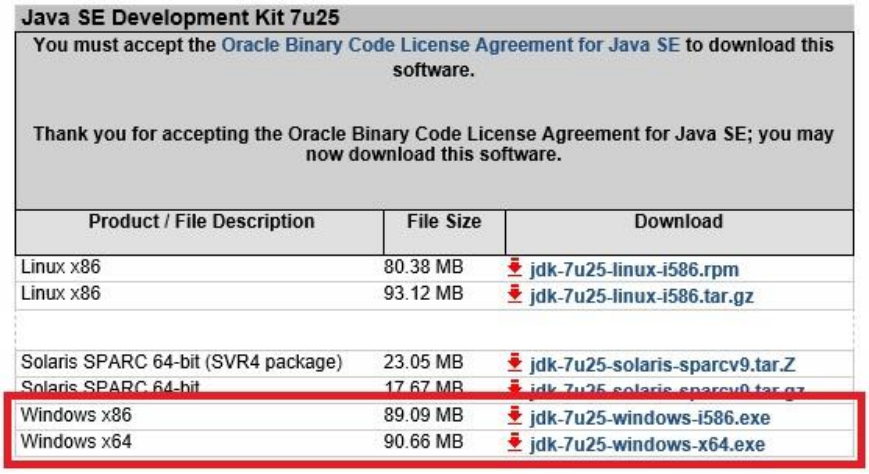
Auf der Befehlszeile gibt man die Anweisung „java -version“ (ein Leerzeichen vor dem Minuszeichen und keines danach) ein und drückt die Eingabetaste *Enter*.

Erscheint nun die Meldung, dass der eingegebene Befehl nicht gefunden wurde, dann ist Java noch nicht installiert. Andernfalls wird die installierte Java-Version angezeigt. Um nun herauszufinden, ob auch das Software Development Kit installiert ist, tippt man „javac“ ein und bestätigt wieder mit Enter. Wenn der Befehl nicht gefunden wurde, ist nur eine Java Runtime-Version installiert, mit der keine eigenen Programme erzeugt werden können.

## JAVA SDK HERUNTERLADEN UND INSTALLIEREN

Falls noch kein Java SDK installiert wurde, muss es aus dem Internet heruntergeladen und eingerichtet werden. Auf Oracles Java-Homepage http://www.oracle.com/technetwork/java/index.html (auch die ehemalige Adresse http://java.sun.com leitet noch hierher weiter) findet man im Bereich „Software Downloads“ einen Link zu „*Java SE*“. „SE“ steht hier für „*Standard Edition*“ und bezeichnet eine Java-Version für normale Anwender und Anwendungsentwickler, in der unter anderem die Komponenten für die Entwicklung und Ausführung von professionellen Unternehmens- und Webanwendungen fehlen, die in der *Enterprise Edition* (*Java EE*) enthalten sind, auf die aber im Rahmen dieser Einführung nicht näher eingegangen wird.

Nach dem Klick auf den Link „Java SE“ erscheint eine Auswahl der Downloads zur aktuellen Java Standard Edition. Wir wählen die Option „JDK“, da wir ein „Java Development Kit“ zur Entwicklung eigener Programme benötigen und klicken hier auf den Button „Download“.



**Java SE Development Kit 7u25**

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Thank you for accepting the Oracle Binary Code License Agreement for Java SE; you may now download this software.

| Product / File Description | File Size | Download |
|---|---|---|
| Linux x86 | 80.38 MB | jdk-7u25-linux-i586.rpm |
| Linux x86 | 93.12 MB | jdk-7u25-linux-i586.tar.gz |
| Solaris SPARC 64-bit (SVR4 package) | 23.05 MB | jdk-7u25-solaris-sparcv9.tar.Z |
| Solaris SPARC 64-bit | 17.67 MB | jdk-7u25-solaris-sparcv9.tar.gz |
| Windows x86 | 89.09 MB | jdk-7u25-windows-i586.exe |
| Windows x64 | 90.66 MB | jdk-7u25-windows-x64.exe |

Downloadlinks der 32- und 64-Bit-Windows-Version des Java Development Kit

Nun scrollen wir zum ersten grau hinterlegten Block mit Downloadlinks, der eine Liste von Downloadmöglichkeiten für das „Java SE Development Kit“ in der aktuellen Version bereitstellt. Hier müssen wir zuerst das Lizenzabkommen

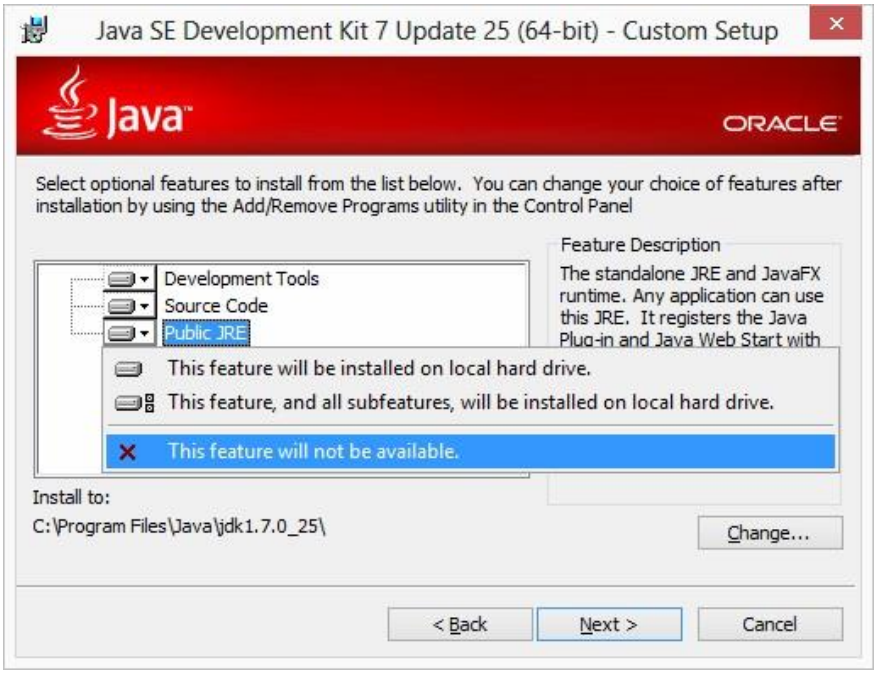
akzeptieren, bevor der Download gestartet werden kann. Nachdem ich davon ausgehe, dass Sie mit Windows arbeiten (worauf auch alle in diesem Buch enthaltenen Beschreibungen ausgerichtet sind), wählen Sie nun einen Link zu einer Windows-Version des SDK. Hier gibt es dann noch zwei Auswahlmöglichkeiten:

1. *Windows x86*: Dies ist die „herkömmliche“ Version, die aktuell noch auf jedem Personal Computer mit einer beliebigen Windows-Version lauffähig ist. Wenn Sie sich nicht sicher sind, ob Sie ein 32-Bit- oder 64-Bit-Betriebssystem verwenden (bei 64 Bit kann beispielsweise mehr Arbeitsspeicher verwendet werden als bei 32 Bit und die Architektur der Recheneinheiten des Computers ist unterschiedlich), wählen Sie diesen Download.
2. *Windows x64*: Eine Java-Version für 64-Bit-Architekturen und 64-Bit-Betriebssysteme. Sie können diese Version probeweise herunterladen; falls sie sich nicht installieren lässt, besitzen Sie kein 64-Bit-Betriebssystem und müssen die x86-Version verwenden.

Nach dem Klick auf die Download-Verknüpfung erscheint unter Umständen ein Dialog, in dem sie gefragt werden, ob Sie die heruntergeladene Datei speichern oder ausführen möchten. Wählen Sie in diesem Fall die Option „Ausführen“.

Nun wird - gegebenenfalls nach einer weiteren zu bestätigenden Sicherheitsabfrage - das Java-Installationsprogramm gestartet.

Nachdem wir den Willkommensbildschirm durch einen Klick auf „Next“ übersprungen haben, bekommen wir nun die zu installierenden Bestandteile des JDK aufgelistet. In den „Development Tools“ sind alle Programme und Werkzeuge enthalten, die wir zum Entwickeln eigener Anwendungen benötigen. Der „Source Code“ wird uns später einmal nützlich dabei sein, die bereits in Java enthaltenen Funktionen besser nutzen und verstehen zu können. Die „Public JRE“ - eine reine Java-Laufzeitumgebung - hingegen ist für unsere Programmierzwecke irrelevant und braucht daher nicht mit installiert zu werden. Dies erreicht man durch einen Klick auf das Symbol links neben „Public JRE“ und die Auswahl „This feature will not be available.“




Installation des Java SDK ohne die Java Runtime (JRE)

Das unter „Install to“ angegebene Verzeichnis sollte man beibehalten. Am besten, Sie schreiben sich den angegebenen Verzeichnispfad auf, denn Sie benötigen ihn später noch.

Ein weiterer Klick auf „Next“ startet nun die Installation. Nach deren Abschluss kann das Installationsprogramm beendet werden. Möglicherweise öffnet sich anschließend noch ein Browserfenster mit weiterführenden Informationen, das Sie aber getrost ignorieren und schließen dürfen. Java ist nun auf Ihrem Computer installiert und muss nur noch für eine einfache Verwendung von überall aus dem Systempfad hinzugefügt werden.

## Das Java-Verzeichnis dem Suchpfad hinzufügen

Wenn Sie auf der Eingabeaufforderung jetzt erneut „java –version“ oder „javac“ aufrufen, werden Sie sich wundern, dass die Ergebnisse dieselben sind wie vor der Installation. Das liegt daran, dass Sie zwar wissen, wohin Sie Java installiert

haben, Ihr Computer aber nicht weiß, wo er das Programm „java“ oder „javac“ finden kann. Sie könnten dieses Problem einerseits dadurch lösen, dass sie dem auszuführenden Programm den vollständigen Verzeichnispfad, in dem es installiert wurde, voranstellen, also beispielsweise:

```
"C:\Program Files\Java\jdk1.7.0_25\bin\javac"
```

Dies würde zum gewünschten Ergebnis führen, nämlich einer Auflistung aller Programmparameter für den Java Compiler. Beachten Sie, dass aufgrund des Leerzeichens im Verzeichnisnamen „Program Files“ die Anführungszeichen um die Eingabe unbedingt erforderlich sind. Um beim Verwenden des Java Compilers nicht immer den vollen Verzeichnispfad angeben zu müssen, empfiehlt sich folgende Lösung: Im Betriebssystem ist ein Suchpfad („PATH“ genannt) definiert, in dem alle Verzeichnisse aufgelistet sind, in denen beim Aufrufen eines Programms nach ebendiesem Programm gesucht werden soll. Wenn wir nun das Verzeichnis *C:\Program Files\Java\jdk1.7.0_25\bin* diesem Suchpfad hinzufügen, können alle darin befindlichen Programme jederzeit aufgerufen werden, egal, in welchem anderen Verzeichnis man sich gerade befindet.

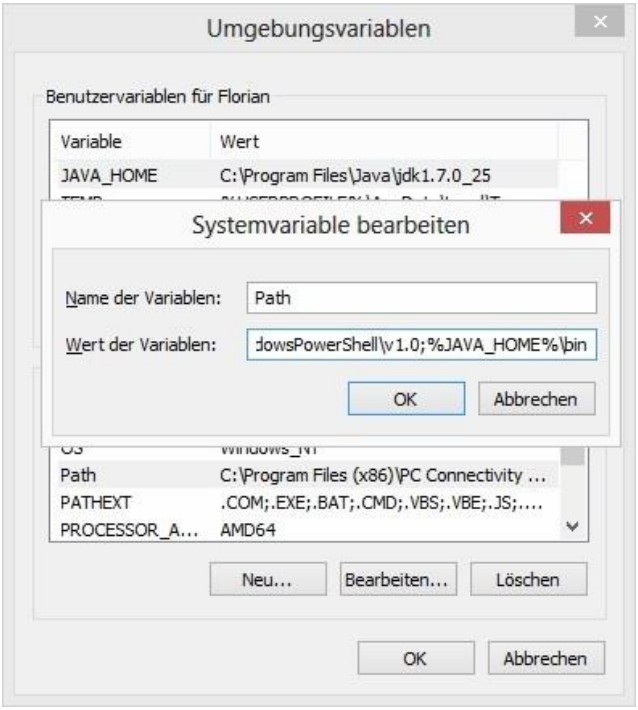
Den Suchpfad kann man über die Einstellungen für die Umgebungsvariablen ändern, die man findet, indem man in der Eingabeaufforderung

```
control sysdm.cpl
```

eintippt und mit Enter bestätigt. In dem Dialog, der sich nun öffnet, wählt man die Registerkarte „Erweitert“ und klickt auf den Button „Umgebungsvariablen…“ Der jetzt erscheinende Dialog bietet die Möglichkeit, Umgebungsvariablen für den aktuellen Benutzer und für jeden Benutzer des Betriebssystems zu verwalten. Zunächst fügen wir eine neue Benutzervariable im oberen Bereich des Dialogs hinzu. Die neue Variable heißt „JAVA_HOME“ und bekommt den Wert „C:\Program Files\Java\jdk1.7.0_25“ (oder das entsprechende Verzeichnis, das Sie sich vorhin aufgeschrieben haben, diesmal ohne Anführungszeichen und ohne das Unterverzeichnis „\bin“ am Ende.

Im unteren Bereich bei den Systemvariablen finden wir bereits die Umgebungsvariable „Path“ in der Liste. Diese klicken wir an und auf den Button „Bearbeiten“. Ganz am Ende der langen Liste von Verzeichnissen, die hier bereits angegeben

sind, fügen wir einen Strichpunkt gefolgt von „%JAVA_HOME%\bin“ (*%JAVA_HOME%* ist ein Platzhalter für das oben in den Benutzervariablen angegebene Java-Verzeichnis) ein.




Konfigurationsdialog für die Umgebungsvariablen

Nach einem Klick auf „OK“ und dem Schließen aller Dialoge ist Java auf Ihrem System nun so eingerichtet, dass es von überall aus verwendet werden kann.

**Hinweis:** In manchen Fällen kann es passieren, dass der Verzeichnis-Platzhalter %JAVA_HOME% nicht richtig aufgelöst wird (Sie merken es daran, dass selbst nach einem Neustart Ihres Computers die Eingabe von „javac“ nicht funktioniert). Schreiben Sie in diesem Fall die Verzeichnisangabe einfach auch vollständig (anstelle des Platzhalters %JAVA_HOME%) an das Ende der Umgebungsvariablen „Path“ (das Unterverzeichnis „\bin“ am Ende nicht vergessen!).

# Das erste selbstgeschriebene Java-Programm

## Anlegen eines Arbeitsverzeichnisses

Damit die selbst entwickelten Java-Programme nicht kreuz und quer auf Ihrer Festplatte herumliegen oder irgendwo im Nirwana verschwinden, empfiehlt es sich, ein eigenes Verzeichnis für alle eigenen Programme und Projekte anzulegen. Dieses Verzeichnis legen wir am besten direkt im Hauptverzeichnis von Laufwerk C: an und nennen es - so ist es unter Java-Entwicklern üblich - „workspace“.

Das Anlegen eines neuen Verzeichnisses lässt sich über den Windows-Explorer oder auch über die bereits geöffnete Eingabeaufforderung erledigen. Egal, in welchem Verzeichnis und auf welchem Laufwerk Sie sich gerade befinden; durch die Eingabe der folgenden Befehle erzeugen Sie das neue Verzeichnis und wechseln auch gleich in dieses:

```
C:                  (Wechseln auf Laufwerk C:)
CD \                (Wechseln ins Hauptverzeichnis)
MD workspace        (Erstellen des Verzeichnisses workspace)
CD workspace        (Wechseln ins Verzeichnis workspace)
```

In diesem Verzeichnis werden wir unsere ersten selbstgeschrieben Programme (und später vielleicht auch einmal ganze Java-Projekte) ablegen.

## Hallo Welt!

In nahezu sämtlicher Fachliteratur und fast jedem Artikel über irgendeine beliebige Programmiersprache wird als Beispiel für den Aufbau und das Aussehen des Programmcodes in dieser Sprache ein „Hallo Welt“-Programm (englisch: „Hello World“) abgedruckt, das nichts weiter macht, als den Text „Hallo Welt“ auf der Befehlszeile auszugeben. Und mit genau solch einem Programm machen auch wir unseren ersten, mutigen Schritt in die Java-Welt.

Den dazu nötigen Programmcode (oft auch *Quellcode* genannt, um ihn vom *Maschinencode*, der ja ebenfalls ein Programmcode ist, zu unterscheiden), schreiben wir in einem beliebigen Texteditor. Am einfachsten ist es - falls sie nicht bereits einen tollen Texteditor auf Ihrem Computer installiert haben - den bei Windows

mitgelieferten Editor *Notepad* zu verwenden. Tippen Sie in der Eingabeaufforderung

```
notepad Hallo.java
```

ein, um den Editor für die neu zu erstellende Quellcode-Datei *Hallo.java* zu öffnen. Jede Java-Quellcode-Datei trägt die Erweiterung „.java“ und beginnt mit einem Großbuchstaben.

Tragen Sie in die neu erzeugte Datei im Editor nun den folgenden Quelltext (ohne die vorangestellten Zeilennummern) ein:

```
1  class Hallo {
2    public static void main(String[] args) {
3      System.out.println("Hallo Welt");
4    }
5  }
```

Auch wenn Sie wahrscheinlich bis auf den Text „Hallo Welt“ im Moment noch überhaupt nichts an diesem Quellcode verstanden haben, speichern Sie nun die Datei *Hallo.java* und wechseln sie zurück zur Eingabeaufforderung.

Hier tippen Sie den Befehl zum Aufrufen des Java-Compilers ein:

```
javac Hallo.java
```

Wenn Sie keinen Tippfehler gemacht haben, erscheint nun ohne Fehlermeldung wieder die Befehlszeile. Der Compiler hat Ihr Quellprogramm in Maschinencode umgewandelt und diesen in einer Datei namens *Hallo.class* im selben Verzeichnis gespeichert. Diese kompilierte Programmdatei führen Sie nun aus, indem Sie

```
java Hallo
```

(diesmal ohne die Erweiterung „.java“ oder „.class“) eintippen und Enter drücken. Wenn die Bildschirmausgabe nun

```
Hallo Welt
```

lautet, haben Sie erfolgreich Ihr erstes eigenes Java-Programm kompiliert und ausgeführt!

## Schrittweise Analyse des Quellcodes

Um ein wenig Licht in jenes Dunkel, das Ihr Verständnis für die gerade programmierten fünf Zeilen noch umgibt, zu bringen, analysieren wir sie im Folgenden Schritt für Schritt.

```
class Hallo {
```

Java ist, wie im Einführungskapitel bereits erwähnt wurde, eine *objektorientierte* Programmiersprache. Das bedeutet, dass man in Java mit sogenannten *Objekten* arbeitet. Eine *Klasse* ist dabei sozusagen die Beschreibung des Objekts und seiner Eigenschaften in Form von Quellcode. Die Begriffe *Klasse* und *Objekt* werden häufig synonym benutzt; eine genaue Unterscheidung der Begriffe *Klasse*, *Objekt* und *Instanz* folgt später noch. Im Moment belassen wir es dabei, dass unsere erzeugte Klasse (daher heißt die kompilierte Klassendatei auch *Hallo.class*) den Namen *Hallo* hat und ihr Name in dieser ersten Zeile definiert wird. Die Quelldatei *Hallo.java* muss dabei (inklusive Groß- und Kleinschreibung) exakt so heißen, wie die darin enthaltene Klasse.

Am Ende der Zeile steht eine öffnende geschweifte Klammer. Sie „umklammert" alles, was ihr folgt bis hin zur letzten schließenden geschweiften Klammer. Das bedeutet: alles, was nach dieser Klammer und vor der letzten geschweiften Klammer (in Zeile 5) kommt, gehört zu dieser Klasse.

```
    public static void main(String[] args) {
```

In dieser Zeile wird die einzige in der Klasse *Hallo* enthaltene Methode definiert. Unter dem Begriff *Methode* versteht man eine Zusammenfassung einzelner Befehle, die gemeinsam einen sinnvollen Zweck erfüllen. Dies wird anhand eines Beispiels aus der realen Welt womöglich verständlicher: Wenn Sie Ihrem Kind sagen, es soll sich die Zähne putzen, meinen Sie genauer gesagt damit in etwa Folgendes:

- Geh ins Badezimmer
- Nimm die Zahnbürste
- Drücke Zahnpasta auf die Bürste
- Schrubbe deine Zähne gründlich für mindestens zwei Minuten
- Spüle deinen Mund und die Zahnbürste gut mit Wasser aus

Die „Methode“ namens *zaehneputzen* umfasst also mehrere einzelne Schritte, die nur alle zusammen und nacheinander in genau dieser Reihenfolge ausgeführt einen Sinn ergeben bzw. einen konkreten Zweck erfüllen. Sie sagen Ihrem Kind auch (nachdem es die einzelnen Schritte gelernt hat) nicht mehr alle fünf Schritte, sondern nur noch quasi den Methodennamen. Anders formuliert rufen Sie bei Ihrem Kind die Methode *zaehneputzen* auf und es erledigt in der Folge die dazu gehörenden Arbeitsschritte selbständig.

Idealerweise kann ich anhand dieses Beispiels zugleich den Unterschied zwischen *Klasse*, *Objekt* und *Instanz* erklären: Ein (abstraktes, also beliebiges und nicht genauer bestimmtes) Kind ist im Beispiel eine *Klasse*. Jedes Kind muss seine Zähne putzen, besitzt bzw. kennt also die Methode *zaehneputzen*. Ihr Kind (falls Sie eines haben, ansonsten nur hypothetisch) wäre dann eine konkrete *Instanz* (also eine Ausprägung) der Klasse *Kind*. Diese konkrete Instanz, die auch einen Namen hat (sei es nun Michael, Clarissa oder was auch immer), ist dann ein *Objekt* (auch wenn diese Bezeichnung für ein Kind ethisch mindestens fragwürdig ist). Kein Kind (im konkreten Sinn als *Instanz* bzw. *Objekt*) ist genau identisch mit irgendeinem anderen, alle haben ihre individuellen Eigenschaften und Fähigkeiten, doch sie alle sind gleichermaßen Kinder (im abstrakten Sinn als *Klasse*) und müssen unter anderem alle ihre Zähne reinigen.

Sie fragen sich vielleicht, wieso ich die Methode *zaehneputzen* mit kleinem Anfangsbuchstaben und mit *ae* statt *ä* geschrieben habe? Bezeichnungen von Methoden oder Funktionen sowie im Allgemeinen auch von Variablen in Java

1. werden immer mit einem kleinen Anfangsbuchstaben geschrieben. Sofern ihr Name aus mehreren Wörtern zusammengesetzt ist, wird der erste Buchstabe jedes weiteren Wortes groß geschrieben, also z. B. *putzDirDieZaehne* (dies nennt man *CamelCase*-Schreibweise),
2. enthalten keine Umlaute oder Sonderzeichen, da dies dazu führen kann, dass die Methode nicht korrekt ausgeführt wird.

Nun aber zurück zur Quellcode-Zeile 2. Die Methode hat den Namen *main*. Dieser Name ist von Haus aus für die Methode vorgesehen, von der ausgehend ein Java-Programm gestartet wird. Im weiteren Programmablauf könnten noch Tausende

andere Methoden und Klassen aufgerufen und ausgeführt werden; seinen Anfang hätte das Programm jedoch stets in einer *main*-Methode genommen.

Das Schlüsselwort *public* bedeutet, dass die Methode für Außenstehende (das können andere Klassen, die unsere Klasse *Hallo* verwenden möchten oder auch wir selbst als diejenigen, welche die Klasse von der Befehlszeile aus ausführen wollen, sein) *sichtbar*, d. h. ansprechbar ist. Würde an dieser Stelle anstatt *public* das Schlüsselwort *private* stehen, könnten wir unsere Klasse nicht starten, da die *main*-Methode dann unsichtbar wäre und von außen nicht aufgerufen werden könnte.

*static* ist ein Schlüsselwort (wie auch schon *public* wird es in der Fachsprache als *Modifier* bezeichnet, da es die Eigenschaften der Methode *modifiziert*), das beschreibt, dass diese Methode keinen Bezug auf eine konkrete Instanz einer Klasse nimmt und damit für alle Instanzen immer das gleiche Ergebnis liefert. Stellen wir uns beispielsweise konkrete Instanzen (Objekte) unserer Klasse *Kind* vor, die allesamt eine Methode *sagDanke* sowie eine Eigenschaft (*Variable*) *muttersprache* besitzen. Bei einer *Kind*-Instanz mit der *muttersprache* Deutsch würde die Methode bzw. Funktion *sagDanke* als Ergebnis „Danke“ zurückliefern. Bei einem Kind mit der *muttersprache* Englisch „Thank you“ und bei einem mit französischer *muttersprache* „Merci“. Solch eine Funktion könnte dann in keinem Fall statisch sein, da sie nicht für jedes Kind dasselbe Ergebnis produzieren darf. Ein augenzwinkerndes Beispiel für eine möglicherweise statische Funktion der Klasse *Kind* wäre wohl *willstDuInsBettGehen*. Hier könnte unabhängig vom konkreten Kind immer „Nein“ zurückgegeben werden.

Der Unterschied zwischen den Begriffen *Methode* und *Funktion* besteht darin, dass eine *Funktion* ein Ergebnis eines definierten Typs zurückliefert und eine *Methode* nicht. Das *void* in Codezeile 2 gibt an, dass von der Methode *main* kein Wert zurückgegeben wird. Würde an dieser Stelle stattdessen z. B. *String* stehen, wäre die *Methode* eine *Funktion* und müsste am Ende eine Zeichenkette (String) zurückliefern. Trotz dieses feinen Unterschieds werden die beiden Begriffe jedoch häufig synonym verwendet. Ich möchte auch selbst nicht garantieren, an jeder Stelle dieses Buchs immer trennscharf zwischen beiden Begriffen zu unterscheiden und bitte Sie insofern gleich einmal um Nachsicht.

Das in runden Klammern hinter dem Methodennamen stehende *String[] args* ist ein *Parameter* der Methode, also eine *Variable* (Platzhalter) für Eingabewerte, die dieser Methode übergeben werden. Im speziellen Fall der *main*-Methode werden in dieser Variable mit dem Namen *args* die Aufrufargumente beim Starten des Programms übergeben. Falls eine Methode keine Parameter hat, stehen hinter ihrem Namen trotzdem die beiden runden Klammern. Nachdem wir beim Aufruf des Programms keine Argumente mit übergeben haben, ist die Variable in diesem Fall zwar überflüssig, muss aber trotzdem immer in der *main*-Methode definiert sein. Das *String[]* vor dem Variablennamen gibt den *Typ* der Variable an. Ein *String* ist einfach eine beliebige *Zeichenkette*, also irgendein Text. Die eckigen Klammern hinter *String* bedeuten, dass in der Variablen *args* nicht nur ein, sondern beliebig viele Strings enthalten sein können. Eine solche Sammlung beliebig vieler gleichartiger Objekte wird als *Array* bezeichnet und gleich noch genauer besprochen. In unserem Fall bedeutet es, dass wir unserer Klasse beliebig viele Aufrufparameter übergeben können. Wir haben jedoch keinen einzigen verwendet. Dies werden wir in der zweiten Ausbaustufe unseres *Hallo*-Programms ändern.

Die geschweifte Klammer am Zeilenende zeigt wiederum den Beginn eines Blocks von zusammengehörenden Anweisungen an. Hier beginnt der sogenannte *Rumpf* der Methode (der in diesem Fall nur eine einzige Anweisung enthält). Durch die schließende geschweifte Klammer in Zeile 4 wird die Methode abgeschlossen.

```
System.out.println("Hallo Welt");
```

In dieser Zeile wird schließlich der eigentliche Zweck der Klasse erfüllt und der Text „Hallo Welt“ auf dem Bildschirm ausgegeben. Die in Java eingebaute Klasse *System* enthält verschiedene statische (also nicht auf eine bestimmte Instanz bezogene) Methoden und Objekte. Ein Punkt dient in Java als Trennzeichen zwischen einem Objekt und den darin enthaltenen Variablen, Objekten und Methoden. *out* ist ein in der Klasse *System* enthaltenes Objekt. Genauer gesagt ist *out* das Objekt, das für die Ausgabe auf dem Bildschirm zuständig ist. Dieses Objekt besitzt nun selbst wieder eine Methode mit dem Namen *println* (für „print line“ = „ drucke Zeile“). Wie schon bei der Deklaration unserer *main*-Methode gesehen, können einer Methode beim Aufruf Parameter (also Werte, mit denen die Methode arbeiten soll) übergeben werden. Diese Parameter stehen immer in

runden Klammern, bei mehreren Parametern sind die einzelnen Parameter innerhalb der Klammern durch Kommas getrennt. Wir übergeben der Methode *println* nun als Parameter einen *String* (erkennbar an den Anführungszeichen), und zwar „Hallo Welt“.

Die Codezeile besagt also:

- Die Klasse *System* wird benutzt,
- aus dieser wird das in ihr enthaltene Objekt *out* verwendet
- und von diesem die Methode *println* aufgerufen,
- die als Aufrufparameter den String „Hallo Welt“ übergeben bekommt.

Der Strichpunkt am Ende der Zeile bedeutet, dass die Anweisung damit zu Ende ist. Direkt danach (oder der Lesbarkeit zuliebe in der nächsten Zeile) könnte die nächste Anweisung folgen. Jede einzelne Anweisung in Java muss stets mit einem Strichpunkt abgeschlossen werden. Dies ermöglicht - hilfreich insbesondere bei langen Anweisungen - auch, die Anweisung über mehrere Zeilen zu verteilen. Unsere Codezeile könnte also z. B. auch so formuliert werden:

```
System  .
  out.
  println
    (    "Hallo Welt"
  )   ;
```

Auch wenn das die Lesbarkeit in diesem Fall nicht erhöht, weiß der Compiler, dass die Anweisung erst nach dem Strichpunkt beendet ist.

Wenn Sie jetzt denken: „Wieso brauche ich fünf Zeilen kryptischen Codes, nur um zwei Wörter auszugeben?“, dann muss ich Ihnen entgegenhalten:

- Die Klassendefinition (Zeilen 1 und 5) ist für jede Klasse (auch für eine winzig kleine Klasse wie unsere) nun einmal nötig. Bei Klassen mit über 100 Zeilen fallen diese zwei Zeilen nicht groß ins Gewicht, bei unseren fünf Zeilen hingegen schon.
- Ebenso braucht eine Klasse, die gestartet werden soll, eine *main*-Methode, deren *Signatur* (Name, Modifier und Parameter) von der Java-Spezifikation vorgegeben und erforderlich ist, damit die Klasse ausgeführt und nicht nur

von anderen Klassen benutzt werden kann. Sobald die Methode einmal mehr als nur eine Zeile Code enthält, fallen die zwei Zeilen für die Methodendeklaration auch nicht mehr so deutlich ins Gewicht.

- Der Code könnte problemlos auf vier oder weniger Zeilen zusammengefasst und sogar in nur einer einzigen Zeile geschrieben werden. Aber wer möchte solchen Code dann noch lesen:

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

An diesem Beispiel sieht man bereits, wie wichtig und sinnvoll es ist, entsprechende Einrückungen zur Verbesserung der Lesbarkeit des Quellcodes zu verwenden. Die Empfehlung lautet hier: Eine öffnende geschweifte Klammer, die den Beginn eines Codeblocks markiert, steht noch am Ende der Zeile, die den Codeblock beschreibt. Die folgenden Zeilen sind jeweils um zwei Leerzeichen weiter eingerückt, bis zur schließenden geschweiften Klammer, die dann wieder zwei Stellen weiter links und somit auf der gleichen Ebene steht, wie die Definition des Codeblocks bzw. der Methode bzw. der Klasse.

Die schließende Klammer in Zeile 4 unseres Codes beendet den Rumpf der Methode *main* und steht auf derselben Ebene (zwei Zeichen eingerückt) wie die Deklaration der *main*-Methode. Die Klammer in Zeile 5 schließt die Klasse *Hallo* ab.

## Einfache Sammlungen gleichartiger Objekte (Arrays)

Nun wollen wir unser *Hallo Welt*-Programm noch geringfügig verändern. Es soll jetzt nicht nur mehr die ganze Welt sondern auch der Anwender, der das Programm aufruft, gegrüßt werden.

Dazu verwenden wir den bisher unbenutzten Aufrufparameter (auch *Argument* genannt) *args* der Methode *main*. Der Typ des Parameters ist wie gesehen *String[]*, also ein *Array* (eine Sammlung/Aufzählung/Liste) von *String*-Objekten (ja, jeder *String* ist zugleich ein eigenes Objekt mit Methoden und Eigenschaften).

Wäre *args* kein *String*-Array, sondern nur ein einzelner *String* (ohne die eckigen Klammern am Ende der Typdefinition), so könnten wir diesen ganz einfach mit der Anweisung

```
System.out.println(args);
```

auf dem Bildschirm ausgeben. Da *args* aber als Array aus beliebig vielen Elementen (die allesamt *String*-Objekte sind) bestehen kann, muss man angeben, welches Element daraus man verwenden möchte. Dies geschieht, indem man hinter die Variable *args* ebenfalls zwei eckige Klammern schreibt und dazwischen den Index (also die Position) des Elements angibt, das man haben möchte. Das erste Element in einer Aufzählung unter Java hat stets den Index 0 (und nicht 1, wie man vermuten würde), das zweite Element den Index 1, das dritte den Index 2 und so weiter. Diese Art zu zählen ist in der Informatik weit verbreitet und war bereits in der Java-Vorgängersprache C an der Tagesordnung. Ein passender Witz dazu lautet: „Man hat schon C-Programmierer gesehen, die im Aufzug auf die 2 gedrückt haben, wenn sie in den dritten Stock wollten."

Wenn wir also nun den ersten in *args* enthaltenen *String* auf dem Bildschirm ausgeben möchten, lautet die passende Anweisung dazu:

```
System.out.println(args[0]);
```

Damit haben wir das Rüstzeug, um die ganze Welt und danach im Speziellen noch den Benutzer des Programms zu begrüßen. Unser Programmcode muss dazu nur um eine weitere Anweisung ergänzt werden:

```
class Hallo {
  public static void main(String[] args) {
    System.out.println("Hallo Welt");
    System.out.println("Hallo " + args[0]);
  }
}
```

Die neue Codezeile gibt wiederum einen *String* aus. Dieser ist aber nun (mit Hilfe des Plus-Zeichens) zusammengesetzt aus der Zeichenkette „Hallo " und dem ersten beim Programmaufruf übergebenen Argument.

Nachdem wir die Änderung gespeichert haben, kompilieren wir den Code mit

```
javac Hallo.java
```

und führen ihn wieder aus, diesmal allerdings mit einem Parameter/Argument:

```
java Hallo Egon
```

Anstelle von „Egon“ kann auch ein beliebiger anderer Name angegeben werden. Die Ausgabe des Programms ist nun:

```
Hallo Welt
Hallo Egon
```

## BEDINGTE AUSFÜHRUNG (*IF*)

Was passiert nun aber, wenn wir (versehentlich oder absichtlich) keinen Namen als Parameter beim Programmaufruf angeben? Versuchen wir es doch einfach mal. Das Ergebnis sieht dann so aus:

```
Hallo Welt
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 0
        at Hallo.main(Hallo.java:4)
```

Die Ausgabe des Strings „Hallo Welt“ hat wie wir sehen noch problemlos funktioniert. Anschließend jedoch ist ein Fehler (in der Java-Sprache eine *Exception*, englisch für *Ausnahme*, was „eine Ausnahme bzw. Abweichung vom üblichen Programmablauf“ bedeutet) aufgetreten. Die Fehlermeldung teilt uns zusätzlich mit, dass der Fehler in der Methode *main*, im Code der Klassendatei *Hallo.java* in Zeile 4 aufgetreten ist und dass es sich dabei um eine *ArrayIndexOutOfBoundsException* beim Zugriff auf den Index 0 handelt.

Wenn wir die Bezeichnung der Exception ins Deutsche übersetzen, erhalten wir „Array-Index außerhalb der Begrenzungen“. Dies gibt uns bereits einen Hinweis auf die Problemursache: Der angegebene Index 0 des ersten Elements des Arrays liegt außerhalb der zulässigen Grenzen, innerhalb derer Index-Werte angegeben werden können. Nachdem wir keinen Aufrufparameter übergeben haben, hat das Array *args* die Länge 0 und es existiert darin eben kein erstes Element mit dem Index 0.

Wenn wir jetzt sicherstellen möchten, dass nicht diese unschöne Fehlermeldung angezeigt wird, wenn wir vergessen haben, beim Aufruf einen Namen als Parameter zu übergeben, müssen wir vor der Ausgabe prüfen, ob *args* auch wirklich (mindestens) ein Element enthält. Anders ausgedrückt müssen wir überprüfen, ob (englisch: *if*) die *Länge* des Arrays *args* größer als 0 ist. Dies erledigt eine sogenannte *if-Abfrage*:

```
class Hallo {
  public static void main(String[] args) {
    System.out.println("Hallo Welt");
    if (args.length > 0) {
      System.out.println("Hallo " + args[0]);
    }
  }
}
```

Nach dem Schlüsselwort *if* ist in Klammern eine Bedingung angegeben. Die Bedingung stellt die Eigenschaft *length* des Arrays *args* (jedes Array besitzt eine Eigenschaft *length*, welche die Anzahl der im Array enthaltenen Elemente angibt) in eine Größer-Beziehung (über das Größer-Zeichen „>“) mit dem Wert 0. Wenn dies der Fall ist, also die Länge von *args* größer als 0 ist, wird mit der öffnenden geschweiften Klammer ein neuer Codeblock betreten. Alle bis zur nächsten schließenden geschweiften Klammer enthaltenen Anweisungen dieses Codeblocks werden ausgeführt, falls die in der Klammer hinter *if* definierte Bedingung zutrifft, ansonsten wird der gesamte Codeblock einfach übersprungen.

Nachdem wir diesen Code nun kompiliert und ausgeführt haben, erscheint die Begrüßung für den einzelnen Anwender dann und nur dann, wenn sein Name beim Aufruf als Parameter übergeben wurde.

## Schleifen über eine Liste von Werten (*for each*)

Die letzte in diesem Kapitel besprochene Erweiterung unseres Programms ermöglicht es uns, gleich mehrere Personen mit Namen zu begrüßen. Wir möchten also nicht mehr nur das erste im Array *args* enthaltene Element verwenden, sondern alle (und zwar beliebig viele) beim Programmstart übergebenen Parameter.

Das hierfür geeignete Programmierkonstrukt nennt sich *for each*-Schleife. *Schleife* deshalb, weil dieselben Verarbeitungsschritte mehrfach hintereinander durchgeführt werden sollen und *for each*, weil die Verarbeitung *für jedes* einzelne Element aus einer Liste von Werten erfolgt. Um die Verständlichkeit des nun schon deutlich umfangreicheren Codes zu verbessern, benennen wir die Variable *args*, die der *main*-Methode übergeben wird, in *namen* um. Dies können wir jederzeit tun, solange der Typ des Parameters (*String[]*) derselbe bleibt. Grundsätzlich ist es immer hilfreich, Variablen „sprechende“ Namen zu geben. Der Standardname der Variable *args* steht für „Argumente“. Da wir jedoch wissen, dass

bei unserem Programm als Argumente ein oder mehrere Namen übergeben werden, taufen wir die Variable auch so.

```
class Hallo {
  public static void main(String[] namen) {
    System.out.println("Hallo Welt");
    if (args.length > 0) {
      for (String name : namen) {
        System.out.println("Hallo " + name);
      }
    }
  }
}
```

Das Schlüsselwort *for* zeigt an, dass nun eine Programmschleife folgt. In der Bedingung ist angegeben, dass aus dem Array *namen* jeder einzelne Wert nacheinander in einem String mit der Bezeichnung *name* gespeichert wird und anschließend der Rumpf der Schleife (wieder begrenzt durch die geschweiften Klammern) jeweils erneut durchlaufen werden soll. Innerhalb des Schleifenrumpfes greifen wir dann auf das jeweils aktuelle Element aus dem Array *namen* über dessen im Schleifenkopf definierte Variablenbezeichnung *name* zu. Verständlicher wird dieses abenteuerliche Codekonstrukt mit dem Doppelpunkt innerhalb der Klammer, wenn man es als „for each (*name* in *namen*)“ ausspricht.

Nachdem wir die Klasse *Hallo.java* nun erneut gespeichert und durch den Java-Compiler in eine .class-Datei umwandeln haben lassen, führen wir das Programm diesmal mit mehreren Aufrufparametern aus:

```
java Hallo Egon Hans Martina Waltraud
```

Die Ausgabe ist jetzt wie folgt:

```
Hallo Welt
Hallo Egon
Hallo Hans
Hallo Martina
Hallo Waltraud
```

Damit haben wir in unserem ersten kleinen Java-Programm bereits mehrere verschiedene Java-Schlüsselwörter und Programmierkonzepte kennen gelernt und

unseren Code schon mit ineinander liegenden Codeblöcken über mehrere Einrückungsebenen tief verschachtelt. Hierbei hat sich noch umso mehr gezeigt, wie wichtig eine korrekte Formatierung des Quellcodes mit den entsprechenden Einrückungen ist, damit man problemlos erkennen kann, wo ein Codeblock beginnt und wo er endet.

Wenn Sie es bis hierher geschafft haben, können Sie dies durchaus als Erfolg verbuchen. Ich habe Sie mit dem erst nachträglich erklärten Quellcode des ersten Hallo Welt-Programms quasi „ins kalte Wasser geworfen“, und Sie haben es geschafft, all die für Sie völlig neuen Dinge zu durchschauen und den Code sogar noch um einige Funktionen zu erweitern. Sie können jetzt bereits bedeutend besser programmieren als die meisten Ihrer Freunde und Bekannten (es sei denn, Sie sind von Informatikern umzingelt).

Um das in diesem Kapitel Gelernte nochmals zu rekapitulieren, folgt nun wie am Ende jedes Abschnitts eine Reihe von Verständnisfragen und Aufgaben, anhand derer Sie Ihren eigenen Wissensstand überprüfen können. Die Lösungen finden Sie im Anhang am Ende dieses Buches.

## Aufgaben zu Kapitel 1

1. Erklären Sie in einem Satz anhand der Begriffe *Klasse* und *Instanz*, was ein *Objekt* ist.

2. Welche der folgenden Aussagen zu Java-Programmen treffen zu? Korrigieren Sie falsche Aussagen.
   a. Java-Quelldateien haben die Dateiendung *.class*.
   b. Die Quelldatei heißt immer genauso wie die darin definierte Klasse.
   c. Der Compiler erzeugt aus den Quelldateien Maschinencode.
   d. Die Quelldateien können direkt ausgeführt werden.
   e. Ein Text in Anführungszeichen ist eine Instanz der Klasse *String*.
   f. Man kann im Quellcode mehrere Strings zu einem zusammensetzen, indem man ein Plus-Zeichen zwischen die Strings schreibt.
   g. Ein Array muss mindestens ein Element beinhalten.
   h. Eine schließende geschweifte Klammer bezeichnet stets das Ende einer Methode.
   i. Ein Punkt im Quellcode trennt die Angabe eines Objekts und einer seiner Methoden oder Eigenschaften/Variablen.
   j. Eine Anweisung muss stets mit einem Punkt abgeschlossen werden.

3. Mit welcher Codezeile könnte man den Text „Hans ist doof!“ auf der Befehlszeile ausgeben, wenn „Hans“ dabei der einzige Aufrufparameter ist, der beim Start des Programms übergeben würde und das Array mit den Aufrufparametern *args* heißt?

4. Wie könnte man erreichen, dass zusätzlich der Text „und Manfred auch!“ ausgegeben wird, falls (und auch nur dann) als zweiter Parameter der Name „Manfred“ angegeben wird?

5. Schreiben Sie eine Codezeile, mit der zudem die Anzahl der beim Programmaufruf übergebenen Argumente ausgegeben wird.

Ende der Leseprobe

Das vollständige Buch „Java für Einsteiger“

(204 Seiten) erhalten Sie über www.amazon.de

oder direkt per E-Mail an java@kottmair.net