Java Programmieren für Einsteiger

Michael Bonacina

2. Auflage: Juli 2018

ISBN 978-3-96645-003-4

Herausgegeben durch:
BMU Media GmbH
Hornissenweg 4
84034 Landshut

Java Programmieren für Einsteiger

Inhaltsverzeichnis

Alle Programmcodes aus diesem Buch sind als PDF zum Download verfügbar. Dadurch müssen Sie sie nicht abtippen:

http://bmu-verlag.de/java-programmieren/

Kapitel 1

Einleitung

Java ist eine Programmiersprache, die in der Praxis ausgesprochen häufig zum Einsatz kommt. Einer der Gründe hierfür besteht in ihren vielfältigen Anwendungsmöglichkeiten. Beispielsweise gibt es zahlreiche Webanwendungen, die in Java programmiert sind. Diese Sprache kommt dabei häufig für die serverseitige Programmierung zum Einsatz. Es ist jedoch auch möglich, die clientseitigen Teile in Java zu programmieren. Früher waren dafür Java-Applets sehr beliebt. Dabei handelt es sich um kleine Programmteile, die in eine HTML-Internetseite eingebunden sind und die die Möglichkeit bieten, dynamische Internetseiten zu erstellen. Sie waren einer der wesentlichen Gründe für den Erfolg dieser Programmiersprache. Mittlerweile kommen sie jedoch nur noch selten zum Einsatz. Java ermöglicht es auch, gewöhnliche Desktop-Anwendungen zu programmieren. Schließlich kommt diese Programmiersprache für Smartphone-Apps für das Betriebssystem Android zum Einsatz. Diese vielfältigen Anwendungsmöglichkeiten zeigen bereits, welche Bedeutung Java im Alltag hat. Diese Programmiersprache zu beherrschen, eröffnet daher viele berufliche Perspektiven. Dieses Buch stellt eine Einführung in Java dar und vermittelt die grundlegenden Eigenschaften dieser Technik. Es richtet sich dabei an Anfänger, die noch keine Programmiererfahrung haben.

1.1 Java – was ist das?

Der Begriff Java hat viele unterschiedliche Bedeutungen. Selbstverständlich kommt vielen Menschen bei diesem Wort zunächst die Insel in Indonesien in den Sinn. Außerdem handelt es sich dabei um die Bezeichnung für eine bestimmte Kaffeebohne. Davon abgesehen spielt Java in der IT eine wichtige Rolle – was der Inhalt dieses Buchs sein

soll. Doch auch in diesem Bereich ist der Begriff Java nicht ganz eindeutig. Dieses Werk befasst sich mit der Programmiersprache Java. Diese ist jedoch Teil der Java-Technologie. Dabei handelt es sich um eine Ansammlung mehrerer Elemente, die alle in Verbindung zueinander stehen.

Die Grundlage der Java-Technologie stellt die Programmiersprache Java dar. Diese besteht aus vielen verschiedenen Befehlen und aus einer fest vorgegebenen Syntax, die die Struktur vorgibt. Damit lassen sich Programme in Java formulieren.

Die Java-Technologie umfasst außerdem das Java Development Kit (JDK). Dieses ist nur für Anwender erforderlich, die selbst in Java programmieren möchten. Dabei handelt es sich um ein Software-Paket, das den Programmcode, bei dem es sich lediglich um Text handelt, kompiliert. Dabei übersetzt er ihn in den sogenannten Java-Bytecode, der die Grundlage eines lauffähigen Java-Programms darstellt. Darüber hinaus sind im JDK verschiedene Bibliotheken enthalten, die vorgefertigte Funktionen für Java-Programme enthalten und dadurch das Programmieren deutlich einfacher gestalten.

Darüber hinaus ist die Java-Laufzeitumgebung von großer Bedeutung. Nach dem englischsprachigen Begriff Java Runtime Environment wird diese auch häufig als JRE abgekürzt. Im Gegensatz zu den beiden anderen Bestandteilen spielt diese nicht nur für Programmierer, sondern auch für die Endanwender eine wichtige Rolle. Dabei handelt es sich um ein System, das die Nutzung von Java-Programmen ermöglicht. Insbesondere die virtuelle Maschine (JVM) ist hierbei von großer Bedeutung. Diese führt das Programm aus. JRE ist für viele unterschiedliche Betriebssysteme erhältlich. Jedes Java-Programm kann darauf ausgeführt werden – unabhängig davon, welches Betriebssystem der Installation zugrunde liegt. Java ist daher plattformunabhängig. Das stellt einen großen Vorteil dieser Technologie dar. Ein Java-Programm läuft problemlos unter Windows, Linux, Unix oder anderen Betriebssystemen – unter der

Voraussetzung, dass die Java-Laufzeitumgebung darauf installiert ist.

1.2 Ein kurzer Überblick über die Geschichte der Programmiersprache

Java entstand zu Beginn der 90er Jahre. 1991 beauftragte der Computer- und Softwarehersteller Sun Microsystems ein Entwicklerteam unter der Leitung von James Gosling mit dem Green Project. Dieses sollte nicht nur eine Programmiersprache umfassen, sondern eine komplette Betriebssystemumgebung mit virtueller CPU. Zu Beginn trug die Technik den Namen Oak (Object Application Kernel) – der Legende nach in Anlehnung an eine Eiche, die vor dem Bürofenster der Entwickler stand. Da jedoch bereits eine Software mit diesem Namen existierte, mussten das Team eine neue Bezeichnung wählen. Es entschied sich für den Namen seiner bevorzugten Kaffee-Sorte – der Java-Bohne. In Anlehnung daran ist das Symbol von Java bis heute eine Kaffeetasse.

Nach nur 18 Monaten Entwicklungszeit präsentierte die Arbeitsgruppe 1992 die erste Java-Anwendung. Dabei handelte es sich um *7 (ausgesprochen als Star Seven) – eine grafische Benutzeroberfläche mit Touchscreen für die Steuerung verschiedener technischer Geräte. Diese Präsentation war ein voller Erfolg und Sun wollte diese Technik für intelligente Haushaltsgeräte verwenden. Daher wuchs das Projekt stark an. Allerdings setzten sich diese Produkte zunächst nicht wie erhofft durch, sodass die Entwicklung kurz vor dem Abbruch stand. Allerdings kam zu dieser Zeit gerade das Internet auf und Sun erkannte sofort das Potenzial, das Java für die Gestaltung dynamischer Internetseiten hatte. Der Durchbruch der Technologie war erreicht, als der Browser Netscape Navigator – zu dieser Zeit einer der beliebtesten Web-Browser – Java integrierte. Damit wurde diese Technologie und mit ihr die zugrundeliegende Programmiersprache zu einem wesentlichen Bestandteil des Internets, das sich zu dieser Zeit immer weiter ausbreitete.

Java übernahm zahlreiche Fähigkeiten und Funktionen von älteren Programmiersprachen. An erster Stelle sind hierbei C und C++ zu nennen. Die Syntax weist sehr große Ähnlichkeiten zu diesen auf. Allerdings war Java von Anfang an als objektorientierte Programmiersprache geplant. (Was das genau ist, wird im Kapitel VII vorgestellt). Aus diesem Grund stellt auch Smalltalk – die erste objektorientierte Programmiersprache – eine wichtige Quelle von Java dar.

1.3 Compiler oder Interpreter – die Besonderheit von Java

Bei einem Computerprogramm handelt es sich zunächst um reinen Text. Dieser enthält alle wichtigen Anweisungen für die Umsetzung und befolgt dabei fest vorgegebene Regeln. Wenn der Anwender dieses Programm nun einfach aufruft, wird ihm aber lediglich der Text angezeigt. Eine Ausführung der Befehle findet nicht statt. Dafür ist ein weiterer Schritt notwendig. Dafür gibt es zwei Möglichkeiten, die beide einige Vor- und Nachteile mit sich bringen. Um die Besonderheiten von Java zu verstehen, ist es wichtig, auf diese Punkte einzugehen.

Um ein Programm auszuführen, ist es notwendig, es in die Maschinensprache zu übersetzen. Dabei handelt es sich um Befehle in Binärcode, die der Prozessor direkt ausführen kann. Bei vielen Sprachen erfolgt die Übersetzung durch einen Compiler. Das ist ein Programm, das die Textdatei, in der das Programm vorliegt, einliest. Es überprüft dabei, ob alle Regeln beachtet wurden. Daraufhin erstellt es eine neue Datei. Diese enthält nun ausschließlich Maschinencode und ist daher direkt ausführbar. Um das Programm auszuführen, ist es lediglich notwendig, die entsprechende Datei anzuklicken.

Darüber hinaus gibt es Interpretersprachen. Die Programme liegen dabei ursprünglich ebenfalls in Textform vor. Für die Übersetzung kommt jedoch kein Compiler zum Einsatz, sondern ein Interpreter. Der Unterschied besteht darin, dass dieser keine ausführbare Datei erzeugt. Er übersetzt das Programm hingegen in die Maschinensprache und lädt daraufhin die Befehle direkt in den Arbeitsspeicher und führt sie

aus. Die Übersetzung erfolgt dabei nur für eine einzelne Anwendung. Wenn das Programm erneut ausgeführt werden soll, muss es aufs Neue übersetzt werden.

Beide Alternativen bieten Vor- und Nachteile. Compilersprachen überzeugen in erster Linie durch eine gute Performance. Da die Übersetzung bereits vor der Ausführung erfolgt, laufen sie sehr schnell ab. Da Interpretersprachen die Übersetzung jedes Mal aufs Neue durchführen, kann es hierbei zu Verzögerungen kommen. Darüber hinaus machen Compilersprachen das Programmieren etwas einfacher, da sie Syntax- und Laufzeitfehler klar trennen. Syntaxfehler sind Verstöße gegen die grundlegenden strukturellen Regeln der Programmiersprache. Laufzeitfehler entstehen, wenn durch die Eingabe spezieller Daten ungültige Werte entstehen – beispielsweise bei einer Division durch 0. Wenn ein Programm nicht richtig funktionieren sollte, macht diese Trennung die Fehlersuche einfacher.

Interpretersprachen überzeugen hingegen durch eine hohe Plattformunabhängigkeit. Eine mit einem Compiler erstellte ausführbare Datei ist immer nur auf einem ganz bestimmten Betriebssystem lauffähig. Bei Interpretersprachen sind normalerweise jedoch Interpreter für viele verschiedene Systeme verfügbar – beispielsweise für Windows, Linux oder MacOS. Auf diese Weise ist es möglich, ein einzelnes Programm auf ganz unterschiedlichen Plattformen auszuführen – selbstverständlich unter der Voraussetzung, dass der entsprechende Interpreter darauf installiert ist.

Nun stellt sich die Frage, in welchen Bereich Java fällt. Diese Programmiersprache zeichnet sich jedoch dadurch aus, dass bei der Umsetzung ein Kompromiss aus beiden Alternativen gewählt wurde. Java verwendet einen Compiler. Dieser übersetzt das Programm jedoch nicht direkt in die Maschinensprache, sondern in den sogenannten Java-Bytecode. Dabei handelt es sich um einen Zwischencode. Der Compiler überprüft die Syntax. Der Bytecode, der dabei entsteht, orientiert sich bereits an der Maschinensprache – was eine spätere Ausführung beschleunigt. Allerdings ist er noch unabhängig von der konkreten Hardware. Dieser Schritt folgt wie bei Interpreter-

sprachen erst bei der Ausführung. Dafür kommt die Java Virtual Machine zum Einsatz. Diese führt den Bytecode schnell und effizient aus. Da diese virtuelle Maschine für viele verschiedene Betriebssysteme erhältlich ist, ist Java plattformunabhängig. Durch diese Kombination verbindet Java zahlreiche Vorteile von Compiler- und Interpretersprachen.

1.4 Java: hervorragend geeignet für den Einstieg in die Programmierung

Aufgrund der großen Anzahl an unterschiedlichen Programmiersprachen fällt es manchmal schwer, sich zu entscheiden, welche davon für den Einstieg gewählt werden soll. Java stellt dabei nur eine von vielen verschiedenen Möglichkeiten dar. Allerdings bietet es gerade für Anfänger viele Vorteile, mit Java zu beginnen. Dieser Abschnitt stellt vor, weshalb es eine gute Wahl ist, diese Programmiersprache für den Einstieg zu wählen.

Ein wichtiger Aspekt für das Erlernen einer Programmiersprache ist sicherlich deren Verbreitung. In diesem Bereich kann Java voll überzeugen. Diese Sprache kommt für unzählige Anwendungen zum Einsatz – von Desktop-Anwendungen bis hin zu Smartphone-Apps. Gerade dieser letzte Punkt zeigt, dass auch sehr moderne und zukunftsträchtige Anwendungen auf Java basieren. Java-Programmierer haben daher sehr umfangreiche Möglichkeiten, um ihre Kenntnisse in die Praxis umzusetzen. Daher bieten gute Java-Kenntnisse auch hervorragende berufliche Perspektiven.

Der vorhergehende Abschnitt machte bereits die Unterschiede zwischen Compiler- und Interpretersprachen deutlich. Für Anfänger ist es sinnvoll, mit einer Compilersprache zu beginnen. Auf diese Weise werden sie beim Kompilieren sofort auf Syntax-Fehler aufmerksam. Das macht es einfacher, ein korrektes und gut funktionierendes Programm zu erstellen.

Variablen kommen in fast jedem Computerprogramm zum Einsatz. Bei den meisten Compilersprachen – so auch bei Java – ist der Umgang mit den Variablentypen recht strikt. Viele Interpretersprachen sind in dieser Beziehung deutlich freier. Um den richtigen Umgang mit den Datentypen zu erlernen, ist ein strikter Umgang jedoch sehr hilfreich. Wenn später eine Programmiersprache erlernt werden soll, die einen freieren Umgang erlaubt, fällt die Umstellung deutlich leichter als umgekehrt.

Zwischen verschiedenen Programmiersprachen gibt es häufig große Ähnlichkeiten. Die meisten von ihnen bauen auf einer älteren Sprache auf und übernehmen einen großen Teil der Syntax derselben. Besonders einflussreich war die Programmiersprache C, die 1972 entstand. Java zählt ebenfalls zu den Programmiersprachen, die auf der C-Syntax aufbauen. Weitere Vertreter sind C++, C#, Perl, PHP, JavaScript und einige mehr. Aufgrund der syntaktischen Ähnlichkeiten ist es relativ einfach, weitere Sprachen aus dieser Familie zu erlernen. Auch aus diesem Grund ist Java reine gute Wahl für die erste Programmiersprache.

Ein weiterer Vorteil ist die Objektorientierung von Java. Dabei handelt es sich um eine spezielle Form der Gestaltung von Programmen, die in einem späteren Kapitel näher erläutert wird. Es kann jedoch schon jetzt vorweggenommen werden, dass es sich dabei um einen der wesentlichen Grundsätze moderner Programmiertechniken handelt. Um sich von Anfang an an diese Vorgehensweise zu gewöhnen, ist es sinnvoll, mit einer objektorientierten Sprache wie Java zu beginnen.

Kapitel 2

Vorbereitungsmaßnahmen für die Programmierung

Um ein Programm mit Java zu erstellen, sind einige Hilfsmittel notwendig. Von besonderer Bedeutung ist dabei das Java Development Kit (JDK). Dieses ist notwendig, um die Programme zu kompilieren und dadurch ihre Ausführung zu ermöglichen. Darüber hinaus ist es wichtig, ein Programm für die Erstellung des Programmcodes zu installieren. Dafür gibt es grundsätzlich zwei Möglichkeiten. Zum einen können die Anwender den Code mit einem Texteditor erstellen. Dieser kennzeichnet bestimmte syntaktische Bausteine des Programms und rückt häufig zusammengehörende Blöcke ein, um die Übersichtlichkeit zu erhöhen. Zum anderen ist es möglich, eine IDE (Integrated Development Environment bzw. integrierte Entwicklungsumgebung) zu verwenden. Diese ermöglicht nicht nur die Eingabe des Programmcodes. Darüber hinaus stellt sie vielfältige Zusatzfunktionen bereit – von der Kompilierung bis hin zum Einfügen vorgefertigter Funktionen. Professionelle Programmierer verwenden in der Regel eine IDE. Deren Nutzung wird auch in diesem Buch vorgestellt. Um die Strukturen von Java zu verstehen, ist es jedoch sinnvoll, zu Beginn den Code vollständig selbst zu schreiben – mit einem Texteditor. Deshalb verwenden die ersten Kapitel des Buchs dieses Werkzeug. Später erfolgt dann die Einführung der IDE. Aus diesem Grund ist es jedoch notwendig, beide Alternativen auf dem Computer zu installieren.

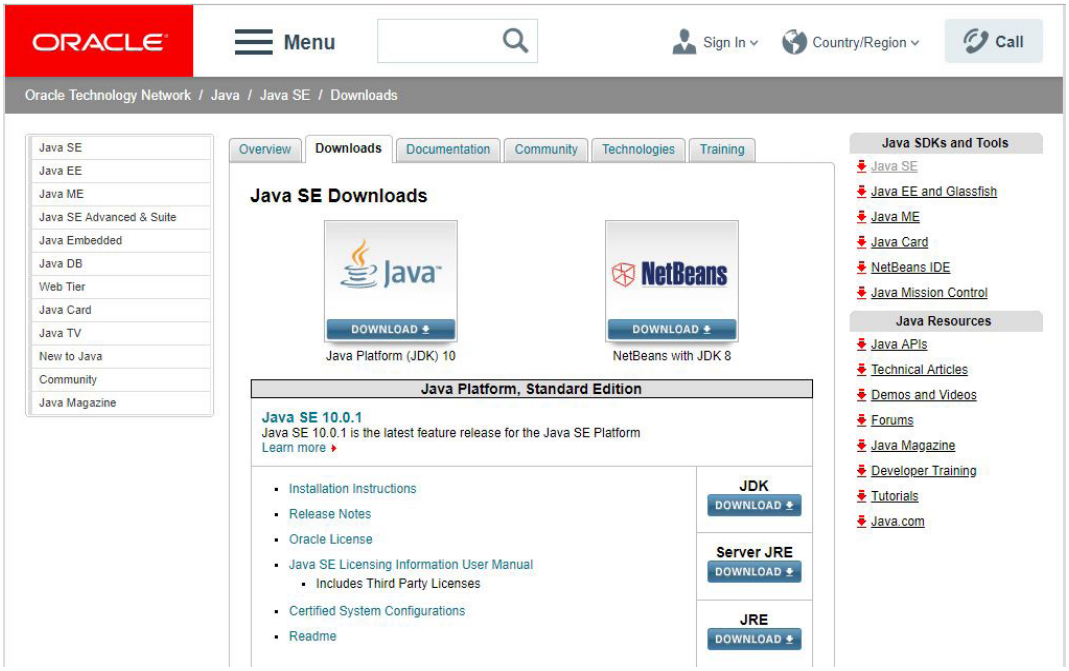
2.1 Java Development Kit und IDE installieren

Um Java-Programme zu erstellen, ist das Java Development Kit unverzichtbar. Dieses enthält unter anderem den Compiler, der den Programmcode in den Java-Bytecode umwandelt und auf diese

Weise die Ausführung des Programms erlaubt. Wie bereits angesprochen, ist auch eine IDE für die Programmierung sehr hilfreich. Hierfür gibt es mehrere Möglichkeiten. Besonders häufig kommen dafür die Programme Eclipse und NetBeans zum Einsatz. Obwohl sich beide Alternativen hervorragend für die Programmierung mit Java eignen, verwendet dieses Lehrbuch die IDE NetBeans. Den Ausschlag dafür gab, dass diese genau wie Java von Oracle entwickelt wird. Das hat unter anderem zur Folge, dass Oracle bei den Java-Tutorials, die das Unternehmen herausgibt, auf diese IDE zurückgreift. Da diese gerade für Anfänger sehr hilfreich sind, ist es sinnvoll, sich von Anfang an in den Umgang mit NetBeans einzuarbeiten. Sowohl JDK als auch NetBeans stehen unter folgender Adresse zum Download bereit:

http://www.oracle.com/technetwork/java/javase/downloads/index.html

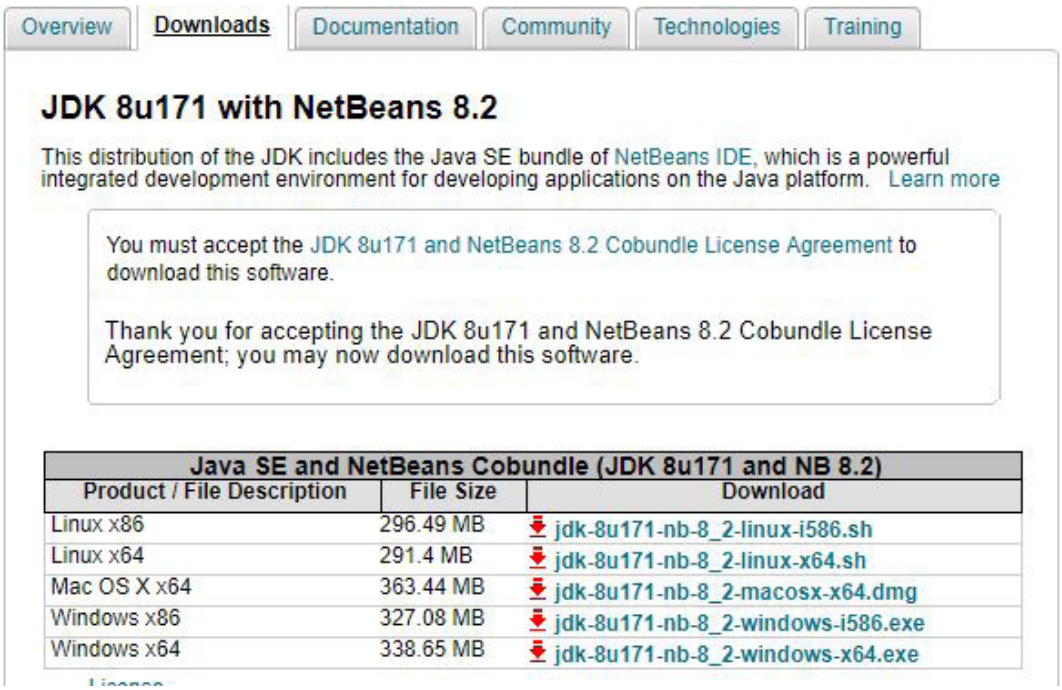
Falls Sie später auch mit Datenbanken arbeiten möchten, beachten Sie bitte auch die Installationshinweise in Kapitel 15.2 bezüglich den verschiedenen Versionen.



Screenshot 1 Die Downloadseite für JDK und NetBeans

Wie auf dem vorhergehenden Screenshot zu sehen ist, besteht die Möglichkeit, JDK separat herunterzuladen oder das Programm zusammen mit der IDE NetBeans zu installieren. Es ist empfehlenswert, die zweite Alternative zu wählen. Das ist nicht nur einfacher. Darüber hinaus stellt diese Vorgehensweise sicher, dass die verwendeten Versionen kompatibel zueinander sind.

Nach dem Klick auf das Symbol für NetBeans wird der Nutzer zu folgender Seite weitergeleitet:



Screenshot 2 Verschiedene Versionen für den Download von JDK und NetBeans

Hier sind verschiedene Versionen von JDK und NetBeans erhältlich. Diese unterscheiden sich in erster Linie durch das Betriebssystem, für das sie vorgesehen sind. Die Software ist für Linux, MacOS und für Windows verfügbar. Darüber hinaus gibt es unterschiedliche Versionen für 32- und für 64-Bit-Rechnerarchitekturen.

Nach dem Download ist es nur noch notwendig, den Installer aufzurufen. Dieser führt den Anwender automatisch durch die Installation. Dabei ist es lediglich erforderlich, einige Angaben zur Installationswei-

se zu bestätigen. In der Regel ist es möglich, die Standardeinstellungen einfach zu übernehmen.

2.2 Das Java Development Kit für die Anwendung vorbereiten

Wenn die Schritte aus dem vorhergehenden Abschnitt durchgeführt sind, ist die Nutzung von JDK bereits möglich. Allerdings nur, wenn die Kompilierung der Programme über die IDE NetBeans stattfindet. Es wurde jedoch bereits dargelegt, dass es für den Anfang sinnvoller ist, die Programme mit einem Texteditor zu erstellen und direkt zu kompilieren. Dafür ist es jedoch in der Regel notwendig, eine weitere Voreinstellung durchzuführen.

Um ein Java-Programm zu kompilieren, ist ein Kommandozeileninterpreter erforderlich. Dieser ist normalerweise bereits in das Betriebssystem integriert und muss daher nicht extra installiert werden. Bei neueren Windows-Versionen sind sogar zwei verschiedene Alternativen vorhanden: Command.exe und Windows Power Shell. Für die in diesem Buch vorgestellten Anwendungen sind beide Programme gut geeignet. Der Anwender erreicht diese beispielsweise über das Windows-Symbol. Anschließend ist es lediglich notwendig, die entsprechenden Programme aus der daraufhin erscheinenden Liste auszuwählen (Command.exe trägt bei deutschsprachigen Windows-Versionen die Bezeichnung Eingabeaufforderung und ist im Ordner Windows-System zu finden). Alternativ dazu ist ein schneller Zugriff über die Windows-Suchfunktion möglich.

Um zu überprüfen, ob JDK bereits nutzbar ist, ist es sinnvoll, folgenden Befehl in den Kommandozeileninterpreter einzugeben:

```
javac -version
```

Wenn das System bereits verfügbar ist, erscheint daraufhin die Anzeige der Version des Java Development Kits. In der Regel erhält der Anwen-

der jedoch folgender Hinweis: “Der Befehl “javac” ist entweder falsch geschrieben oder konnte nicht gefunden werden.”

```
Eingabeaufforderung
Microsoft Windows [Version 10.0.16299.371]
(c) 2017 Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\PC>javac -version
Der Befehl "javac" ist entweder falsch geschrieben oder
konnte nicht gefunden werden.

C:\Users\PC>
```

Screenshot 3 Die Fehlermeldung im Kommandozeileninterpreter

Der Grund für diesen Fehler liegt normalerweise nicht darin, dass JDK falsch installiert ist. Durch den Wechsel in den Ordner, in dem das System installiert wurde, ist es möglich, die Version anzeigen zu lassen:

```
Eingabeaufforderung
C:\>cd program files\java\jdk1.8.0_171\bin

C:\Program Files\Java\jdk1.8.0_171\bin>javac -version
javac 1.8.0_171

C:\Program Files\Java\jdk1.8.0_171\bin>
```

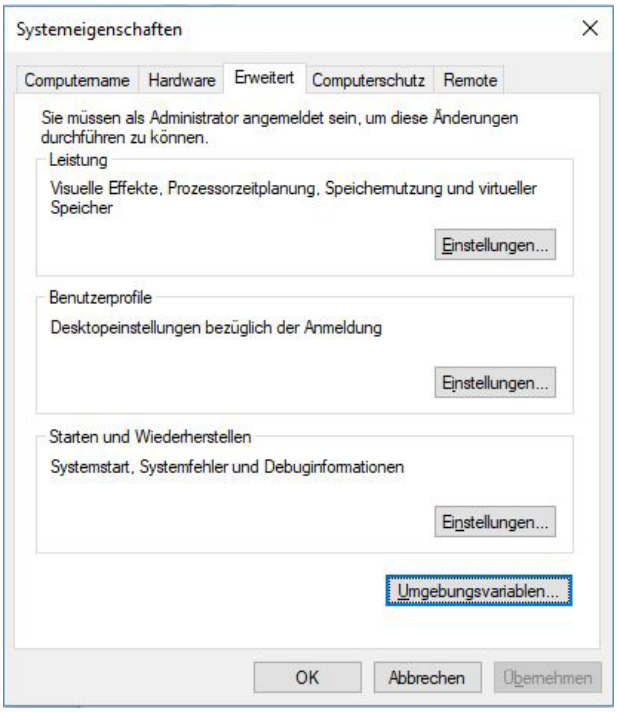
Screenshot 4 Die Anzeige der Version im Verzeichnis, in dem sich die Software befindet

Dafür ist es zunächst notwendig, ins Stammverzeichnis zu wechseln. Dafür ist der Befehl cd .. so oft erforderlich, bis der Kommandozeilen-

interpreter nur noch die Linie C:\>anzeigt. Danach folgen der Befehl cd und anschließend der Pfad, in dem das Programm installiert wurde. Daran wird der Ordner bin angefügt. Häufig ist dabei folgende Eingabe notwendig: cd program files\java\jdk1.8.0_171\bin. Allerdings kann sich der Pfadname auch unterscheiden – je nachdem, welche Einstellungen bei der Installation ausgewählt wurden und um welche JDK-Version es sich handelt.

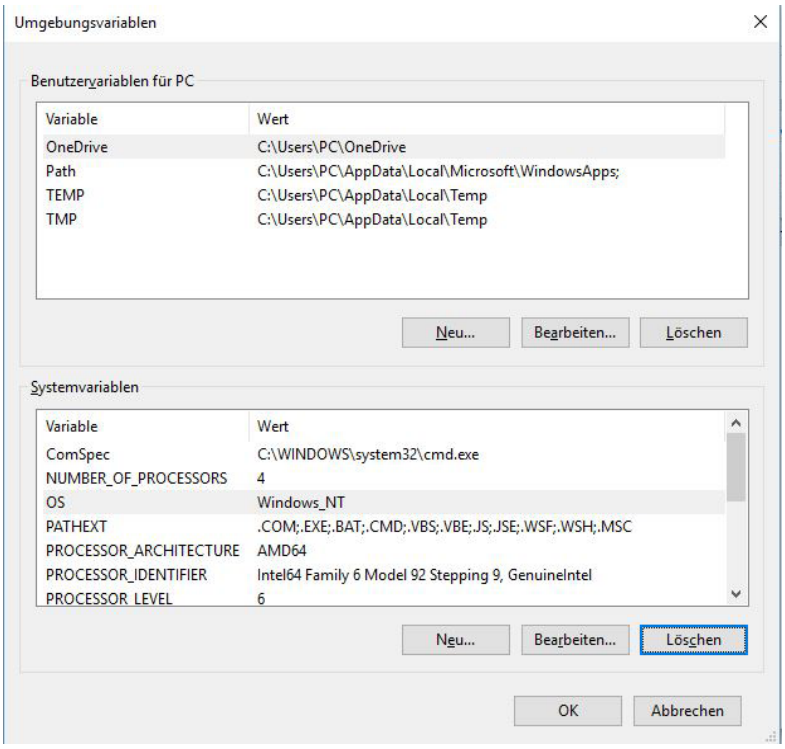
Dieser kleine Test zeigte, dass JDK zwar bereits installiert ist, dass der Zugriff darauf jedoch nur aus dem entsprechenden Ordner möglich ist. Da es nicht sinnvoll ist, die eigenen Programme im gleichen Ordner wie die Software selbst abzulegen, ist es wichtig, die Funktion auch in anderen Bereichen zugänglich zu machen. Dies geschieht durch die Erstellung einer Umgebungsvariablen.

Unter Windows ist es hierfür notwendig, die erweiterten Systemeinstellungen aufzurufen. Das geschieht entweder über das Windows-Symbol und anschließend durch einen Klick auf das Zahnrad am linken Rand. Alternativ ist es auch hierfür sehr praktisch, die Suchfunktion zu verwenden. Anschließend sollte folgendes Fenster erscheinen:



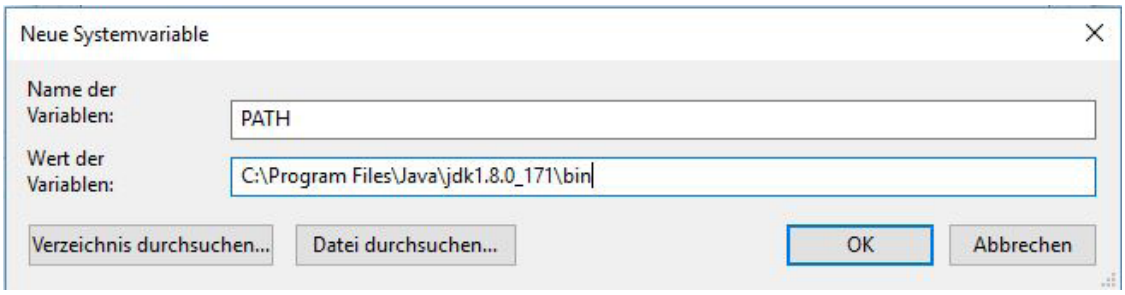
Screenshot 5 die erweiterten Systemeigenschaften

Im unteren Bereich des Fensters erscheint ein Button mit der Beschriftung Umgebungsvariablen. Ein Klick darauf führt zu folgendem Fenster:



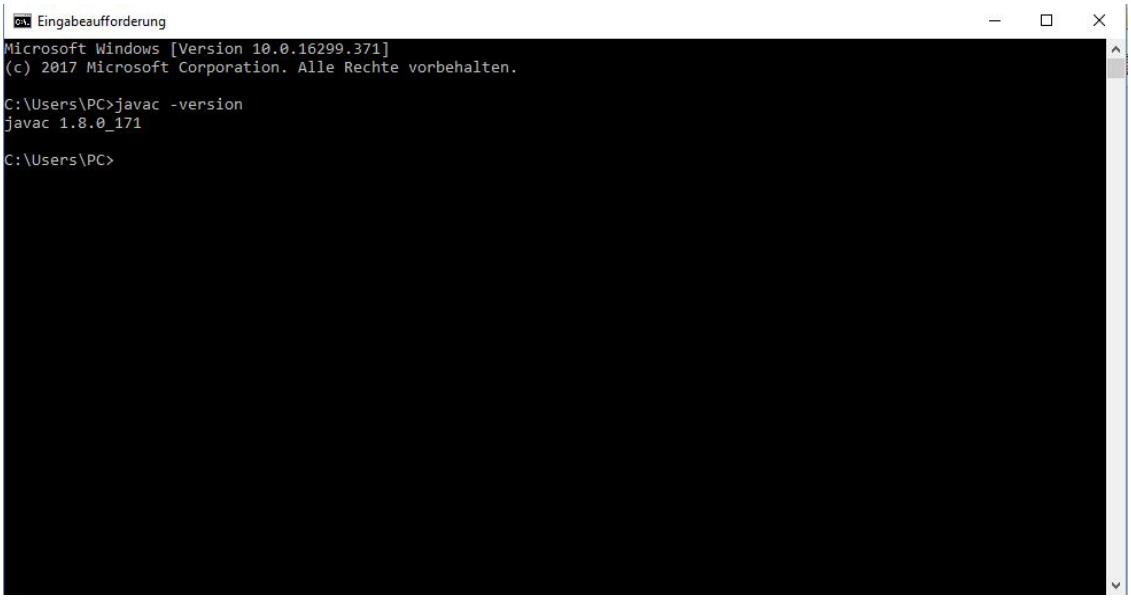
Screenshot 6 Die Umgebungsvariablen

Für die Einstellung ist das untere Feld mit der Bezeichnung Systemvariablen von Bedeutung. Wenn hier noch kein Eintrag mit dem Name PATH vorhanden ist, ist es notwendig, auf den Button "Neu" zu klicken. Im folgenden Fenster ist es nun erforderlich, den Pfadnamen einzugeben. Dieser hängt davon ab, in welchem Verzeichnis JDK installiert wurde. In diesem Beispiel ist der Pfadname C:\Program Files\Java\jdk1.8.0_171\bin.



Screenshot 7 eine neue Umgebungsvariable hinzufügen

Sollte bereits ein Eintrag mit dem Namen PATH vorhanden sein, ist es notwendig, diesen zu überarbeiten. In diesem Fall ist es erforderlich, an den bisherigen Eintrag ein Semikolon und anschließend den entsprechenden Pfadnamen anzuschließen. Sowohl bei der Neugestaltung als auch bei der Änderung treten die Einstellungen nur in Kraft, wenn beide Fenster durch einen Klick auf die Schaltfläche OK bestätigt werden. Danach muss der Kommandozeileninterpreter geschlossen und erneut aufgerufen werden. Nun sollte der Zugriff auch aus anderen Ordnern möglich sein. Zur Überprüfung dient wiederum die Versionsabfrage:



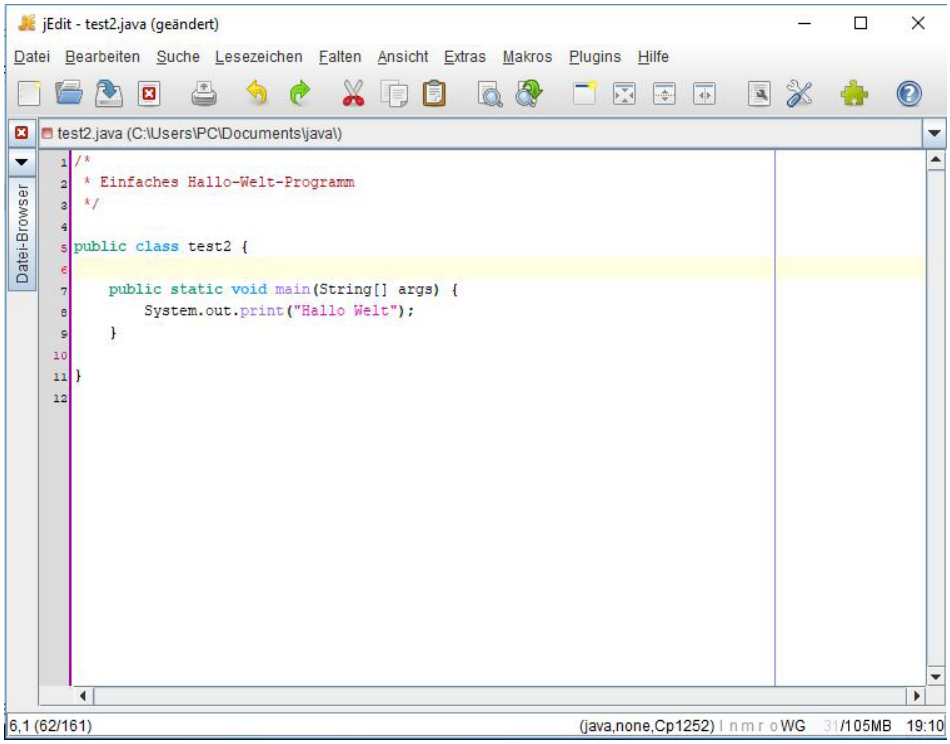
Screenshot 8 Der Befehl javac ist nun auch in anderen Ordnern verfügbar.

2.3 Einen passenden Texteditor installieren

Beim Code eines Java-Programms handelt es sich um reinen Text. Um diesen zu erstellen, ist ein passendes Programm notwendig. Die naheliegendste Lösung wäre es wohl, ein gängiges Textverarbeitungsprogramm wie Word dafür zu verwenden. Das ist jedoch nicht möglich. Derartige Programme fügen dem eigentlichen Text noch viele weitere Informationen hinzu – beispielsweise die Schriftart, die Größe und die Farbe. All diese Zusatzinformationen wären beim Pro-

grammieren störend. Daher ist es notwendig, hierfür einen Texteditor zu verwenden. Dieser zeichnet sich dadurch aus, dass er nur den Text abspeichert.

Bei vielen Betriebssystemen ist bereits eine geeignete Software installiert. Die Linux-Distribution Ubuntu verwendet beispielsweise gedit. KDE nutzt zu diesem Zweck den Texteditor Kate. Beide Alternativen eignen sich bereits hervorragend zum Programmieren, sodass kein weiterer Editor installiert werden muss. Auch Windows verfügt bereits über eine integrierte Software für diese Aufgabe. Diese trägt den Namen Microsoft Editor. Sie ist jedoch auch unter der früheren Bezeichnung Notepad bekannt. Damit ist das Programmieren zwar möglich, allerdings handelt es sich dabei um eine sehr einfache Ausführung. Hochwertige Texteditoren verfügen beispielsweise über eine farbige Syntaxhervorhebung, die bestimmte Schlüsselbegriffe markiert. Sie rücken zusammengehörige Textblöcke automatisch ein und außerdem ermöglichen sie es, einzelne Bereiche, die im Moment nicht benötigt werden, einzuklappen. Das macht den Programmcode übersichtlicher. Darüber sind in der Regel viele weitere nützliche Funktionen integriert.



Screenshot 9 farbliche Syntax-Hervorhebung in einem Texteditor

Der Microsoft Editor bieten diese Funktionen jedoch nicht an. Daher ist es empfehlenswert, einen anderen Texteditor zu installieren. Dafür kommen viele verschiedene Möglichkeiten infrage. Unter folgendem Link ist eine Übersicht über die verschiedenen Alternativen verfügbar: https://de.wikipedia.org/wiki/Texteditor .

Im Prinzip kommt jeder der hier aufgeführten Editoren für die Gestaltung der Programme infrage. Dieses Lehrbuch verwendet den Texteditor jEdit. Dieser steht unter http://www.jedit.org/ zum Download bereit. Die Software ist kostenfrei erhältlich und unterstützt die Programmierung in Java. Außerdem ist die Software selbst in Java programmiert und stellt daher auch ein Anwendungsbeispiel für diese Programmiersprache dar. Der Leser kann sich jedoch frei entscheiden und auch einen anderen Texteditor für die Aufgaben in diesem Buch verwenden.

Kapitel 3

Das erste Programm gestalten

Nachdem die Vorbereitungsmaßnahmen abgeschlossen sind, ist es nun an der Zeit, das erste Programm mit Java zu erstellen. Dessen Funktion soll ausgesprochen einfach sein. Es soll lediglich einen kleinen Text auf dem Bildschirm ausgeben. Obwohl die Funktionen sehr begrenzt sind, ist es dabei möglich, die grundlegenden Strukturen eines Java-Programms kennenzulernen. Um mit dieser Programmiersprache vertraut zu werden, ist es empfehlenswert, den hier vorgestellten Programmcode selbst in den Texteditor einzugeben und anschließend das Programm zu kompilieren und auszuführen. Um ein guter Programmierer zu werden, ist viel Übung notwendig. Daher ist es sinnvoll, bereits bei diesen kleinen Beispielen zu beginnen, auch wenn sie sehr einfach sein mögen.

3.1 Ein Programm mit einer einfachen Ausgabe schreiben

Das erste Programm soll eine einfache Nachricht auf dem Bildschirm ausgeben. Passend zu diesem Lehrbuch soll diese lauten: "Herzlich Willkommen zum Java-Kurs!"

Um eine derartige Ausgabe zu erzeugen, gibt es in Java mehrere Möglichkeiten. Beispielsweise kann der Programmierer hierfür ein eigenes Fenster öffnen oder sogar eine grafische Benutzeroberfläche gestalten. Für das erste Programm kommt jedoch der Befehl System.out.print zum Einsatz. Dieser gibt den Text einfach im Kommandozeileninterpreter aus. Das ist zwar nicht besonders ansprechend, doch ist die Funktionsweise besonders einfach und daher für Anfänger leicht zu verstehen.

Um die Ausgabe zu erzeugen, ist es lediglich notwendig, den entsprechenden Befehl in das Programm einzufügen und danach den ge-

wünschten Text zu schreiben. Dieser muss jedoch in runden Klammern stehen. Das ist notwendig, damit klar wird, auf welchen Bereich sich der entsprechende Befehl bezieht. Darüber hinaus ist es notwendig, den Text in Anführungszeichen zu setzen. Dabei kommen nicht die deutschen Anführungszeichen zum Einsatz, bei denen der öffnende Teil nach unten gestellt ist. Anstatt dessen sind hierbei beide Anführungszeichen im oberen Bereich zu setzen. Der Texteditor erledigt dies jedoch in der Regel ganz automatisch. Die Anführungszeichen sind notwendig, um kenntlich zu machen, dass es sich hierbei um einen Text handelt. Fehlen die Anführungszeichen, weiß das Programm nicht, wie es mit dem Inhalt umgehen soll – es könnte sich dabei beispielsweise auch um Variablen handeln. Das führt zu einer Fehlermeldung. Um den Befehl abzuschließen, ist ein Semikolon notwendig. Die vollständige Programmzeile lautet demnach:


```
System.out.print("Herzlich Willkommen zum Java-Kurs!");
```

Damit ist zwar der entsprechende Befehl erstellt, doch ist das Programm damit noch nicht vollständig. Hierfür sind weitere Bestandteile notwendig. Diese rahmen die einzelnen Befehle ein und verleihen dem Programm daher eine passende Struktur. Ohne sie ist das Programm nicht lauffähig. Der vollständige Code lautet daher:

```
public class Willkommen {

    public static void main(String[] args) {
        System.out.print("Herzlich Willkommen zum Java-Kurs!");
    }
}
```

Das mag auf den ersten Blick etwas kompliziert erscheinen, doch ist der Aufbau im Prinzip ganz einfach und bei allen Programmen gleich. Die Bedeutung der einzelnen Bestandteile wird im nächsten Abschnitt erklärt. Zuvor ist es wichtig, das Programm abzuspeichern. Dafür soll der Name Willkommen.java gewählt werden.

3.2 Die verschiedenen Elemente des Programmcodes

Der eigentliche Befehl, der eine bestimmte Aktion innerhalb des Java-Programms durchführt, wurde bereits im vorherigen Kapitel erläutert. Um einfache Programme zu schreiben, ist dies eigentlich ausreichend. Es ist lediglich notwendig, die übrigen Programmelemente hinzuzufügen, ohne deren genaue Funktion zu kennen. Durch das Hinzufügen oder Ändern von Befehlen lassen sich auf diese Weise bereits viele interessante Programme erstellen. Die Elemente, die diese Kommandos einrahmen, ändern sich dabei nicht. Dennoch ist es sinnvoll, sie bereits jetzt zu erklären. Auf diese Weise versteht der Leser deren Funktion und damit den grundsätzlichen Aufbau von Java-Programmen. Wenn später kompliziertere Prozesse programmiert werden, bei denen auch die grundlegenden Programmelemente abgeändert werden, ist es dann einfacher, die hierfür notwendigen Schritte zu verstehen.

Um die Bedeutung der einzelnen Teile zu erklären, ist es sinnvoll, den Programmcode nochmals im Texteditor anzuschauen. Dabei entsteht folgende Anzeige:

```
1 public class Willkommen {
2
3     public static void main(String[] args) {
4         System.out.print("Herzlich Willkommen zum Java-Kurs!");
5     }
6 }
```

Screenshot 10 Die Anzeige des Programmcodes im Texteditor

Anmerkung: In anderen Texteditoren kann sich die farbliche Gestaltung unterscheiden. Jedoch sollten dabei die gleichen Schlüsselbegriffe hervorgehoben werden. Bei manchen Programmen passiert dies jedoch erst, nachdem der Code mit der Endung .java abgespeichert wurde.

Die farbliche Hervorhebung macht bereits deutlich, dass Begriffe wie `public`, `class` oder `main` eine besondere Bedeutung haben. Besonders wichtig ist dabei das Schlüsselwort `class`. Jedes Java-Programm ist

in Klassen organisiert. Um mit einem Programm zu beginnen, ist es daher notwendig, eine Hauptklasse zu öffnen. Darüber hinaus ist es jedoch möglich, viele weitere Klassen in das Programm zu integrieren.

Auch der Begriff `public` ist sehr wichtig. Bei jeder Klasse ist es möglich, diese als `public` oder als `private` zu definieren. Auf private Klassen kann nur aus der Klasse zugegriffen werden, in der sie sich selbst befinden. Das schützt die Daten vor einem unberechtigten Zugriff. Der Zugriff auf öffentliche Klassen kann hingegen von jedem beliebigen Programmteil und sogar von außerhalb des Programms stattfinden. Da es sich hierbei um die Hauptklasse handelt, findet der Zugriff immer von außerhalb statt. Daher ist es notwendig, sie als `public` zu definieren.

Anschließend folgt der Name der Klasse. Diesen kann der Programmierer relativ frei wählen. Dafür kann er Buchstaben, Zahlen, den Unterstrich (_) und das Dollarzeichen ($) verwenden. Verboten sind alle übrigen Sonderzeichen und spezielle in Java reservierte Schlüsselbegriffe (Ein Beispiel hierfür ist der Begriff `public`, der bereits vorgestellt wurde. Dieser wäre nicht als Klassenname geeignet.) Der Bezeichner darf nicht mit einer Ziffer beginnen und außerdem keine Leerzeichen enthalten. Es ist üblich, für den Namen der Klasse ein Nomen zu verwenden und ihn mit einem Großbuchstaben zu beginnen. Das ist zwar nicht zwingend vorgeschrieben, es verbessert jedoch die Übersichtlichkeit. Sehr wichtig ist es, dass der Name der Hauptklasse stets dem Dateinamen entspricht. Sonst ist es nicht möglich, das Programm zu kompilieren. Wenn wie in diesem Beispiel der Name der Hauptklasse "Willkommen" lautet, muss das Programm unter dem Namen Willkommen.java gespeichert werden.

Der Inhalt der Klasse steht immer in geschweiften Klammern (`{}`). Java-Programme verwenden einzelne Blöcke – beispielsweise Klassen, Methoden oder Schleifen. Diese sind häufig ineinander geschachtelt. Um deutlich zu machen, wo ein Block beginnt und wo er endet, kommen geschweifte Klammern zum Einsatz.

Darauf folgt die Methode `main`. Diese stellt den Startpunkt dar, an dem der Java-Interpreter mit der Ausführung des Programms beginnt. Sie darf nur ein einziges Mal in jedem Programm vorkommen. Ihre Inhal-

te stehen ebenfalls in geschweiften Klammern. Der Interpreter arbeitet die einzelnen Befehle dann in der Reihenfolge ab, wie sie innerhalb der main-Methode aufgeführt sind.

Was der Modifikator `public` bedeutet, wurde bereits erklärt. In diesem Fall kommt ein weiterer Modifikator – `static` – hinzu. Um dessen genaue Funktionsweise zu erklären, wäre es notwendig, etwas tiefer auf die objektorientierte Programmierung einzugehen. Das geschieht jedoch erst in einem späteren Kapitel. An dieser Stelle sei nur so viel gesagt, dass dieser Modifikator notwendig ist, um von außerhalb auf eine Methode zugreifen zu können.

Anschließend folgt der Begriff `void`, der ebenfalls einer Erklärung bedarf. Wenn eine Methode aus einem anderen Programmteil aufgerufen wird, ist es möglich, dass sie einen Wert zurückgibt. Die Methode `main` dient jedoch nur dazu, das Programm zu starten. Ein Rückgabewert ist daher nicht erforderlich. Funktionen ohne Rückgabewert werden mit dem Begriff `void` gekennzeichnet.

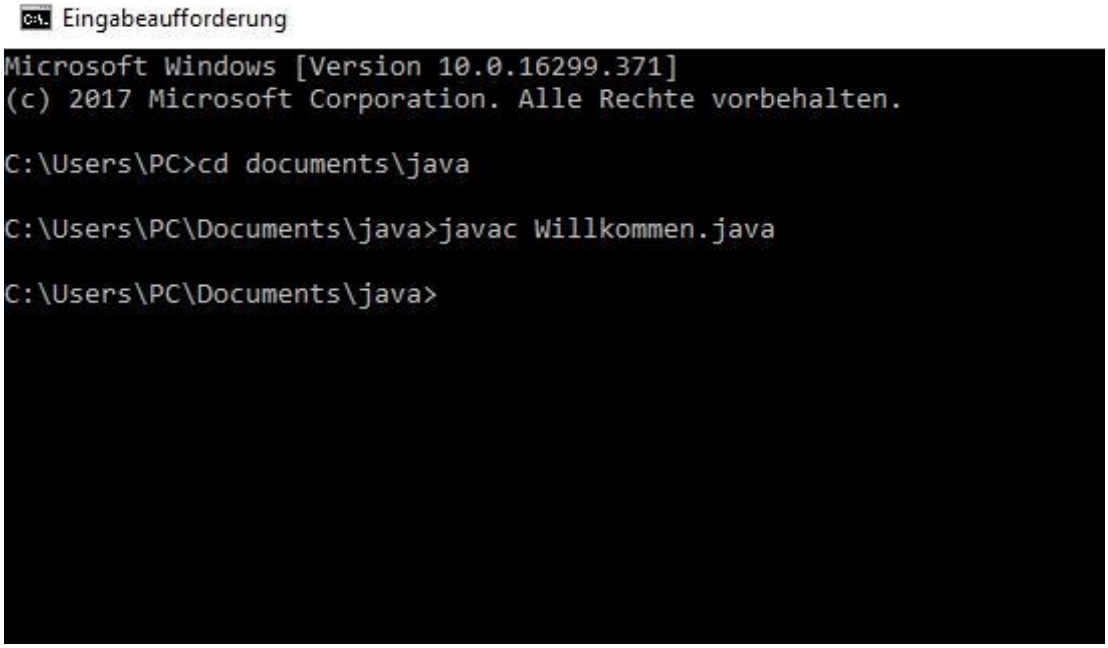
Schließlich bleibt noch zu klären, was es mit dem Ausdruck `(String[] args)` auf sich hat. Es ist möglich, einer Methode verschiedene Parameter zu übergeben. Auf diese kann sie zugreifen und ihre Werte weiterverarbeiten. In diesem kleinen Programm ist dies nicht notwendig, weshalb die entsprechenden Werte nicht weiter beachtet werden. Dennoch ist es notwendig, diesen Ausdruck in die `main`-Methode zu integrieren, da es sonst nicht möglich ist, das Programm zu kompilieren. In manchen Programmen kommt anstatt dessen der Ausdruck (`String... args`) zum Einsatz. Dieser ist vollkommen gleichbedeutend.

3.3 Das Programm kompilieren und ausführen

Bislang wurde zwar ein Programm im Texteditor erstellt, doch kam dieses noch nicht zur Ausführung. Wie dies abläuft, soll der folgende Abschnitt vorstellen. Wie bereits in der Einleitung erwähnt, ist es notwendig, den Programmcode dafür zunächst zu kompilieren. Diese Aufgabe findet wieder über den Kommandozeileninterpreter statt. Dafür ist es

erforderlich, in das Verzeichnis zu wechseln, in dem das Programm abgespeichert wurde. Das ist über den Befehl cd und die Eingabe des entsprechenden Pfadnamens möglich. Die Kompilierung findet dann mit dem Befehl javac statt – gefolgt vom Namen der Datei. In diesem Beispiel ist es daher notwendig, folgenden Code in den Kommandozeileninterpreter einzugeben:


```
javac Willkommen.java
```



Screenshot 11 So wird das Programm kompiliert

Nach der Eingabe des entsprechenden Befehls ist zunächst keine Änderung zu erkennen. Nach einigen Sekunden Wartezeit zeigt der Kommandozeileninterpreter wieder genau die gleiche Eingabeaufforderung wie zuvor an. Lediglich wenn ein Fehler beim Programmieren unterlaufen ist, erscheint eine Fehlermeldung. Wenn der Code jedoch sorgfältig abgeschrieben wurde, sollte das in diesem Fall nicht passieren.

Allerdings hatte die Eingabe dieses Befehls Auswirkungen, die erst auf den zweiten Blick zu erkennen sind. Das wird bei der Betrachtung des entsprechenden Ordners deutlich. Dafür ist es möglich, wie gewohnt

das Verzeichnis über Windows aufzurufen. Eine Alternative dazu stellt es dar, einfach den Befehl dir in den Kommandozeileninterpreter einzugeben. Dieser listet den Inhalt des aktuellen Verzeichnisses auf. Dabei wird deutlich, dass eine neue Datei entstanden ist, die vor der Kompilierung noch nicht existierte. Diese trägt den Namen Willkommen.class. Dabei handelt es sich um die kompilierte Datei, die den Java-Bytecode enthält. Diese kann nun durch die virtuelle Maschine ausgeführt werden.

Zu diesem Zweck ist es notwendig, den Befehl java einzugeben. Darauf folgt der Dateiname – in diesem Fall jedoch ohne Endung. Daraufhin führt die JVM das Programm aus. Nun erscheint der Text, der für die Ausgabe programmiert wurde, im Kommandozeileninterpreter.

Screenshot 12 Die Ausgabe des ersten Java-Programms

3.4 Kommentare für ein leichteres Verständnis des Programms

Programmiersprachen haben die Eigenschaft, dass sie für den Menschen nicht immer leicht zu verstehen sind. Ein erfahrener Programmierer erkennt zwar sofort die verschiedenen Anweisungen und deren Wirkung. Allerdings ist es oftmals nicht klar, welchen Zweck diese erfüllen sollen. Häufig wird das erst aus einem größeren Kontext klar. Bei um-

fangreichen Programmen ist es jedoch sehr langwierig und schwierig, den kompletten Programmcode zu lesen, um den Kontext zu verstehen. Das kann in einigen Fällen große Probleme bereiten – beispielsweise wenn der Programmierer nach einiger Zeit eine Funktion des Programms abändern will und sich nicht mehr genau an die Strukturen erinnert oder wenn ein anderer Programmierer eine Erweiterung hinzufügen will.

Wie fast alle Programmiersprachen enthält Java jedoch ein Instrument, um den Code verständlicher zu machen: Kommentare. Diese machen es möglich, zu erklären, wozu ein bestimmter Programmteil dient. Das macht es später deutlich einfacher, die Funktion zu verstehen. Eine entsprechende Kennzeichnung führt jedoch dazu, dass diese Kommentare keinerlei Einfluss auf die Ausführung des Programms haben. Sie dienen lediglich dazu, den Quellencode übersichtlicher zu gestalten. Es ist sinnvoll, sich bereits von Anfang an die Verwendung von Kommentaren anzugewöhnen.

Für einzeilige Kommentare kommt der Doppelslash (`//`) zum Einsatz. Der Compiler ignoriert alle Inhalte, die in einer Zeile nach diesen Symbolen stehen. Diese Form der Kommentare kommt in der Regel für kurze Anmerkungen zu einer spezifischen Funktion des Programms zum Einsatz. Für längere Kommentare empfiehlt es sich, die Kombination aus `/*` und `*/` zu verwenden. Alle Inhalte, die zwischen diesen beiden Symbolen stehen, werden als Kommentar behandelt – selbst wenn sie sich über mehrere Zeilen erstrecken.

Eine besondere Form von Kommentaren steht zwischen den Symbolen `/**` und `*/`. Dabei handelt es sich um einen Dokumentationskommentar. Dieser erscheint nicht nur im Quellencode, sondern wird durch das Programm javadoc weiterverarbeitet und ist – falls eine entsprechende Dokumentation erstellt wird – auch für den Anwender sichtbar. Er steht direkt vor Klassen, Attributen oder Methoden.

Um die Funktionsweise des ersten Programms leichter verständlich zu machen, ist es möglich, verschiedene Kommentare einzufügen. Der Programmcode könnte beispielsweise wie folgt aussehen:

```
/*Dieses ist das erste Programm, das im Java-Buch erstellt wird.
Es dient dazu, eine einfache Nachricht auf dem Bildschirm auszugeben.
Außerdem soll es die grundlegende Struktur eines Java-Programms ver-
deutlichen.
*/

/**
Ausgabe einer einfachen Nachricht
*/

public class Willkommen {

    public static void main(String[] args) {
        //In der main-Methode steht der Befehl für die Ausgabe
        //der Nachricht.
             System.out.print("Herzlich Willkommen zum Java-Kurs!");
    }
}
```

3.5 Übungsaufgabe: Mehrere Zeilen mit einem Java-Programm ausgeben

Um das Erlernte zu vertiefen, steht am Ende der meisten Kapitel eine kleine Übungsaufgabe. Diese ist in der Regel mit den Kenntnissen, die in den vorherigen Abschnitten vermittelt wurden, zu lösen. Sollten dabei neue Befehle notwendig sein, die noch nicht erwähnt wurden, werden diese kurz erklärt.

Nach der Aufgabenstellung wird eine Musterlösung angegeben. Es ist jedoch empfehlenswert, immer zuerst zu versuchen, das Programm selbst zu gestalten, ohne die Lösung zu betrachten. Sollten dabei Fehler auftreten, werden diese beim Kompilieren angezeigt. Auf diese Weise lernt der Leser, mit diesen Fehlermeldungen umzugehen. Wenn er später eigene Programme ohne Musterlösung entwerfen will, hat er daher ein wichtiges Werkzeug zur Hand, um eventuelle Fehler zu beheben. Erst wenn das Programm läuft oder wenn es selbst nach vielen Versuchen nicht möglich war, es zum Laufen zu bringen, sollten die Musterlösungen betrachtet werden.

1. Schreiben Sie ein Programm, das mehrere Textzeilen auf dem Bildschirm ausgibt. Um einen Zeilenumbruch zu erzeugen, ist es sinnvoll,

anstatt `System.out.print` den Befehl `System.out.println` zu verwenden. Auf diese Weise stehen die darauf folgenden Inhalte in einer neuen Zeile.

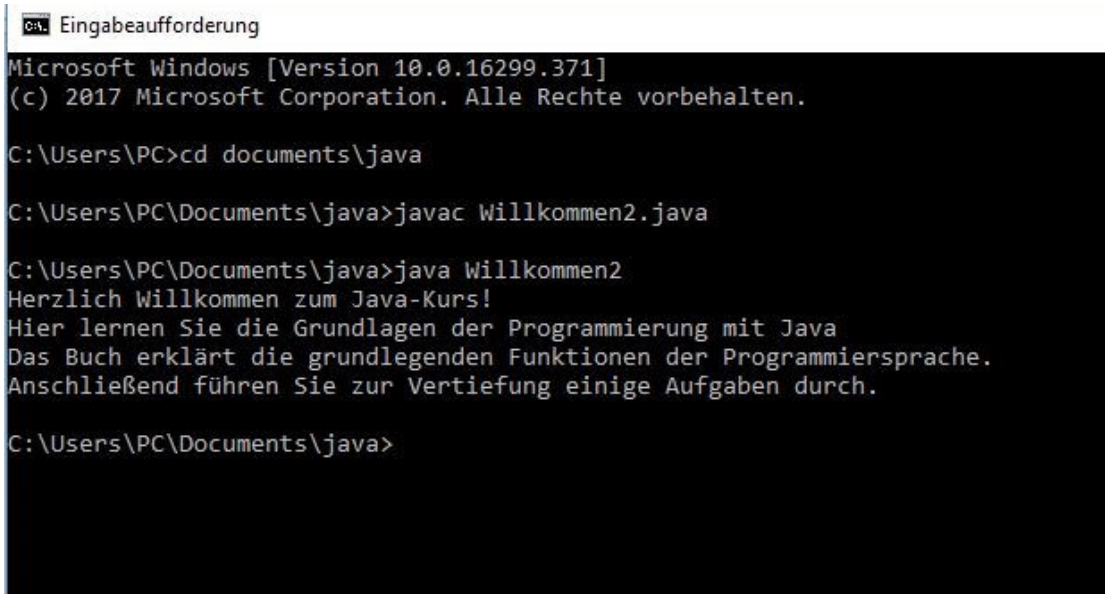
2. Fügen Sie in das eben erstellte Programm Kommentare ein, die das Verständnis erleichtern.

Lösungen:

1.

```
public class Willkommen2 {

    public static void main(String[] args) {
        System.out.println("Herzlich Willkommen zum Java-Kurs!");
        System.out.println("Hier lernen Sie die Grundlagen
                der Programmierung mit Java");
        System.out.println("Das Buch erklärt die
                grundlegenden Funktionen der Programmiersprache.");
                System.out.println("Anschließend führen Sie zur
                Vertiefung einige Aufgaben durch.");
    }
}
```



Screenshot 13 Kompilierung und Ausgabe des Programms

2. Das Programm mit Kommentaren:

```
/*Dieses Programm gibt mehrere Textzeilen aus.
Dafür kommt der Befehl System.out.println zum Einsatz.
*/

/**
Programm für die Ausgabe mehrerer Textzeilen
*/
public class Willkommen2 {

   public static void main(String[] args) {
      //Jede Textzeile wird mit System.out.println erzeugt.
        System.out.println("Herzlich Willkommen zum Java- Kurs!");
        System.out.println("Hier lernen Sie die Grundlagen
                der Programmierung mit Java");
        System.out.println("Das Buch erklärt die
                grundlegenden Funktionen der Programmiersprache.");
          System.out.println("Anschließend führen Sie zur
                Vertiefung einige Aufgaben durch.");
    }
}
```

Hat Ihnen diese Leseprobe gefallen?

Bestellen Sie das Buch als Taschenbuch komfortabel auf Amazon!

eBook: https://amzn.to/2GIsOep

Taschenbuch: https://amzn.to/2SYOQQk

Möchten Sie über neue Bücher und Sonderangebote vom BMU Verlag informiert werden?

Tragen Sie sich jetzt in unseren E-Mail Newsletter ein: https://bmu-verlag.de/