Java-Kompendium

Professionell Java
Programmieren Lernen

Markus Neumann

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

www.bmu-verlag.de
support@bmu-verlag.de

Lektorat: Matthias Kaiser
Einbandgestaltung: Pro ebookcovers Angie
Druck und Bindung: Wydawnictwo Poligraf sp. zo.o. (Polen)

Taschenbuch-ISBN: 978-3-96645-053-9
Hardcover-ISBN: 978-3-96645-054-6
E-Book-ISBN: 978-3-96645-052-2

Java-Kompendium

Inhaltsverzeichnis

Downloadhinweis

Alle Programmcodes und Schaltpläne aus diesem Buch stehen kostenfrei zum Download bereit. Dadurch müssen Sie Code nicht abtippen.

Außerdem erhalten Sie die eBook Ausgabe zum Buch im PDF Format kostenlos auf unserer Website:

Kapitel 1
Einleitung

Gute Kenntnisse im Bereich der Informatik werden immer wichtiger. Diese spielen zum einen eine wichtige Rolle, um sich in einer Welt mit immer weiter voranschreitender Digitalisierung zurechtzufinden. Zum anderen haben sie einen erheblichen Einfluss auf den beruflichen Erfolg. Gute Programmierkenntnisse gewinnen in vielen verschiedenen Berufen an Bedeutung und stellen immer häufiger eines der zentralen Einstellungskriterien dar. Es kommt hinzu, dass sich Berufe im Bereich der Informatik häufig durch eine sehr gute Vergütung auszeichnen. Daher lohnt es sich auch in finanzieller Hinsicht, das Programmieren zu erlernen.

Um sich die entsprechenden Kenntnisse anzueignen, gibt es mehrere Möglichkeiten. Eine davon stellt dieses Buch dar. Es vermittelt umfassende Programmierkenntnisse von Grund auf. Dafür sind keinerlei Vorkenntnisse notwendig. Es gibt Fachkräften verschiedener Berufsgruppen die Möglichkeit, sich weiterzubilden. Außerdem stellt es einen guten Einstieg dar, wenn man später ein Studium oder eine Ausbildung im IT-Bereich aufnehmen will. Auch wer sich gerne in der Freizeit mit diesem Thema befasst, findet hier alle notwendigen Informationen. Dieses Buch vermittelt alle Kenntnisse, die für die Gestaltung fortgeschrittener Programme notwendig sind..

Wer mit dem Programmieren beginnen will, muss sich zunächst für eine Programmiersprache entscheiden. Dieses Buch befasst sich mit der Sprache Java. Diese zeichnet sich durch umfangreiche Anwendungsmöglichkeiten aus. Früher kam Java in erster Linie für die Gestaltung dynamischer Internetseiten zum Einsatz. Allerdings hat die Verwendung in diesem Bereich stark abgenommen. Aktuell liegt der Schwerpunkt bei der Gestaltung von Desktop-Anwendungen und Smartphone-Apps. Java eignet sich jedoch noch für einige weitere Aufgaben hervorragend. Diese umfangreichen Anwendungsmöglichkeiten führten dazu, dass sich Java zu einer der am häufigsten verwendeten Programmiersprachen entwickelte. Daher ist diese Sprache bestens geeignet, um mit dem Programmieren zu beginnen.

1.1 Die Entstehung von Java

Die Programmiersprache Java entstand zu Beginn der 90er Jahre. Das heißt, dass es sich hierbei weder um eine ganz neue noch um eine alte Programmiersprache handelt. Sie ist bereits gut etabliert, doch erfüllt sie auch die Anforderungen der modernen Informatik. Das führt dazu, dass viele Programmierer sie gerne für ihre Projekte verwenden.

Die Entstehung geht auf das sogenannte Green Project zurück. 1991 beauftragte der bekannte Softwarehersteller Sun Microsystems den Programmierer James Gosling damit, ein Team zusammenzustellen, um eine neue Programmiersprache zu entwickeln. Darüber hinaus sollte für die Ausführung der entsprechenden Programme eine vollständige Betriebssystemumgebung entstehen, die auch eine virtuelle CPU umfasste. Dieses Projekt erhielt zunächst die Bezeichnung Oak – als Abkürzung für Object Application Kernel. Da es sich hierbei um den englischen Ausdruck für Eiche handelt, gibt es auch die Legende, dass die Programmierer diesen Namen aufgrund eines Baumes wählten, der von ihrem Büro aus zu sehen war. Allerdings kam es bald darauf zu einer Namensänderung, da ein anderes Unternehmen bereits die Rechte für diesen Begriff innehatte. Das Team wählte als neue Bezeichnung Java. Diese geht nicht auf die asiatische Insel mit dem gleichen Namen zurück, sondern auf den Lieblingskaffee der Entwickler: die Java-Bohne. Das ist daran ersichtlich, dass es sich beim Logo der Programmiersprache um eine Kaffeetasse handelt.

Hinsichtlich der Syntax orientiert sich Java sehr stark an der Programmiersprache C. Dabei handelt es sich um eine der ältesten noch heute verwendeten Programmiersprachen. Viele Funktionen und Fähigkeiten sind dabei identisch. Ein großer Unterschied zu C besteht jedoch darin, dass Java eine objektorientierte Programmiersprache ist. Was das genau bedeutet, wird später in diesem Buch noch erklärt. Daher stellt auch die erste objektorientierte Programmiersprache – Smalltalk – eine wichtige Grundlage von Java dar.

Nach etwa 18 Monaten Entwicklungszeit stellten die Entwickler bereits die erste Java-Anwendung vor. Diese trug die Bezeichnung *7, was als Star Seven ausgesprochen wird. Dabei handelte es sich um eine grafische Benutzeroberfläche, mit der sich per Touchscreen verschiedene technische Geräte steuern ließen. Das Ziel bestand darin, diese Anwendung für die Steuerung intelligenter Haushaltsgeräte zu verwenden. Obwohl die Präsentation ein voller Erfolg war und Sun das Entwicklerteam aufstockte, kam es schon bald darauf zu Problemen. Da die Entwicklung intelligenter Haushaltsgeräte nicht wie geplant voranschritt, war auch kein System für die Steuerung notwendig. Das Projekt stand deshalb bereits kurz vor dem Aus.

Allerdings gewann zu dieser Zeit das Internet immer mehr an Bedeutung. Sun erkannte, dass Java auch hierfür ein enormes Potenzial bot. Daher änderte sich der Verwendungszweck deutlich: Java diente fortan in erster Linie der Erstellung dynamischer Internetseiten. Der Durchbruch war erreicht, als der Netscape Navigator, bei dem es sich damals um den am weitesten verbreiteten Webbrowser handelte, die Java-Technologie implementierte. Java wurde daher zu einer der wesentlichen Programmiersprachen im Bereich der Webentwicklung. Mit dem Siegeszug des Internets stieg auch deren Verwendung deutlich an.

1.2 Die Java-Technologie: Programmiersprache, Compiler und Laufzeitumgebung

Der Begriff Java steht in der Informatik für verschiedene Elemente: für die Programmiersprache Java, die dieses Buch vorstellt, für das Java Development Kit (JDK) und für die Java-Laufzeitumgebung (JRE – Java Runtime Environment). Alle drei Elemente gehören jedoch zusammen. Gemeinsam stellen sie die Java-Technologie dar. Dieser Abschnitt soll daher kurz vorstellen, welche Aufgabe die einzelnen Elemente haben und wie das Zusammenspiel aussieht.

Im Zentrum dieses Buchs steht die Programmiersprache Java. Daher soll diese an erster Stelle vorgestellt werden. Hierbei handelt es sich zum einen um eine Ansammlung verschiedener Befehle. Zum anderen gibt die Sprache feste Regeln für den Aufbau eines Programms vor. Diese werden als Syntax bezeichnet. Anhand der Befehle und der Syntax-Regeln kann man ein Java-Programm schreiben. Dabei handelt es sich jedoch zunächst ausschließlich um ein Dokument in Textformat.

Das JDK ist ein Softwarepaket, das ausschließlich für Java-Entwickler notwendig ist. Gewöhnliche Anwender müssen dieses nicht installieren, um ein Java-Programm auszuführen. Die Aufgabe dieses Pakets besteht darin, den Programmcode aufzubereiten, sodass daraus später ein lauffähiges Programm entsteht.

Die Java-Laufzeitumgebung ist der dritte Bestandteil der Java-Technologie. Diese ist erforderlich, um ein Java-Programm auszuführen. Daher muss sie jeder Anwender auf seinem Computer installieren. Da Java jedoch sehr weit verbreitet ist, zählt JRE zum Standard und ist auf den meisten Betriebssystemen bereits vorinstalliert. Daher ist es in der Regel nicht notwendig, sich darum zu kümmern. Das ist die Grundlage für eine breite Verwendung. Die Ausführung übernimmt die Java Virtual Machine (JVM). Dieses Paket ist für alle gängigen Betriebssysteme erhältlich.

Um die Besonderheiten der Programmiersprache Java zu verstehen, ist es notwendig, kurz auf die allgemeine Funktionsweise eines Computerprogramms einzugehen. Wenn man dieses schreibt, besteht es zunächst nur aus Text. Beim Anklicken der entsprechenden Programmdatei wird diese noch nicht ausgeführt. In der Regel öffnet sich in diesem Fall ein Texteditor, der den entsprechenden Quelltext anzeigt.

Damit es zu einer Ausführung kommt, ist es notwendig, das Programm zunächst in die Maschinensprache zu übersetzen. Dabei handelt es sich um eine Abfolge einzelner Befehle, die an den Prozessor übermittelt werden, damit dieser sie ausführt. Diese bestehen wiederum aus einer Abfolge einer bestimmten Anzahl einzelner Bits. Wie viele Bits pro Befehl enthalten sind, hängt von der Prozessorarchitektur ab. Heutzutage sind 64-Bit-Prozessoren üblich. Das bedeutet, dass jeder Befehl eine Länge von 64 Bit aufweist. Bei älteren Rechnern sind die Werte jedoch deutlich geringer. Daraus wird bereits ersichtlich, dass die Maschinensprache stets von der Prozessorarchitektur abhängt. Doch auch das verwendete Betriebssystem spielt eine Rolle. Das

hat zur Folge, dass der Übersetzungsprozess stets an diesen Faktoren ausgerichtet sein muss.

Wenn man nun das Programm ausführen will, hat man zwei verschiedene Möglichkeiten, um diesen Übersetzungsprozess durchzuführen. Eine Alternative besteht darin, das Programm zu kompilieren. Das bedeutet, dass man nach der Fertigstellung des Programmcodes diesen mit einer Software, die als Compiler bezeichnet wird, übersetzt. Auf diese Weise entsteht ein ausführbares Programm in Maschinensprache. Wenn man die entsprechende Datei anklickt, wird sie unmittelbar ausgeführt.

Die andere Alternative besteht darin, einen Interpreter zu verwenden. Auch diese Software übersetzt den Programmcode in die Maschinensprache. Allerdings erzeugt sie dabei keine ausführbare Datei. Die Befehle werden sofort nach der Übersetzung an den Prozessor übermittelt und nicht gespeichert. Das bedeutet, dass der Übersetzungsprozess jedes Mal aufs Neue stattfindet, wenn man das Programm ausführt.

Welche dieser beiden Alternativen zum Einsatz kommt, kann man nicht frei wählen. Das hängt immer von der verwendeten Programmiersprache ab. Beide Möglichkeiten bringen einige Vor- und Nachteile mit sich. Ausführbare Programme bringen den Vorteil mit sich, dass sie bereits in übersetzter Form vorliegen und somit direkt ausgeführt werden können. Das führt neben einer hohen Nutzerfreundlichkeit zu einer hohen Effizienz. Interpretierte Programme müssen hingegen jedes Mal aufs Neue übersetzt werden. Das beeinträchtigt die Performance deutlich. Insbesondere bei Programmen mit umfangreichen Rechenprozessen wirkt sich das nachteilig aus.

Es wurde bereits angesprochen, dass die Maschinensprache stets von der Rechnerarchitektur und vom Betriebssystem abhängt. Das bedeutet, dass ein ausführbares Programm, das für einen bestimmten Rechner erstellt wurde, nur auf gleichartigen Systemen ausführbar ist. Wenn man beispielsweise ein anderes Betriebssystem verwendet, wäre es notwendig, das Programm neu zu kompilieren. Für den Endanwender, der diese Aufgabe normalerweise nicht ausführen kann, bedeutet dies, dass die Verwendung hier nicht möglich ist. Interpretierte Programme bestehen jedoch lediglich aus dem Programmcode. Die Übersetzung übernimmt der Interpreter, der stets auf das verwendete System ausgerichtet ist. Voraussetzung für die Verwendung ist lediglich, dass der Anwender diese Software installiert hat. Das stellt jedoch normalerweise kein Problem dar, da sie für alle gängigen Programmiersprachen in Versionen für unterschiedliche Betriebssysteme verfügbar ist. Das bedeutet, dass interpretierte Programme plattformunabhängig sind.

Java wählt in dieser Hinsicht einen Mittelweg und versucht dabei, die Vorteile beider Alternativen so gut wie möglich zu nutzen. Wenn man ein Java-Programm schreibt und daraufhin ausführen will, muss man es kompilieren. Der Compiler übersetzt es jedoch nicht in die Maschinensprache, sondern in eine Zwischensprache – den sogenannten Java-Bytecode. Dieser ist der Maschinensprache bereits sehr ähnlich. Platt-

formspezifische Anweisungen werden hierbei jedoch noch allgemein gehalten. Diese werden erst im Rahmen der Interpretierung durch JRE hinzugefügt. Daher kann man das entsprechende Programm auf verschiedenen Betriebssystemen verwenden, sodass man vom wesentlichen Vorteil einer interpretierten Sprache profitiert. Wenn die Java-Laufzeitumgebung das Programm dann ausführt, ist jedoch bereits der größte Teil des Übersetzungsprozesses erledigt. Es sind nur noch wenige plattformspezifische Details zu erledigen. Daher ist dafür deutlich weniger Rechenleistung notwendig als bei gewöhnlichen interpretierten Sprachen. Die Effizienz ist deshalb beinahe die gleiche wie bei einem kompilierten Programm.

1.3 Programmieren lernen mit Java

Wenn man gerade mit dem Programmieren beginnt, ist es notwendig, hierfür eine passende Programmiersprache auszuwählen. Diese Entscheidung hat einen großen Einfluss darauf, wie schnell man dabei Fortschritte erzielt und welche Möglichkeiten später für die Anwendung bestehen. Daher soll dieser Abschnitt kurz darauf eingehen, weshalb es sinnvoll ist, das Programmieren mit Java zu lernen.

Ein wesentlicher Punkt, der für Java spricht, ist die große Verbreitung dieser Sprache. Hierbei handelt es sich um eine der am weitesten verbreiteten Programmiersprachen weltweit. Es ist möglich, viele verschiedene Anwendungen mit Java zu programmieren – von Desktop-Programmen für den PC bis hin zu Smartphone-Apps. Gerade der letzte Punkt zeigt auch, dass Java für ganz neue Entwicklungen zum Einsatz kommt. Daher handelt es sich hierbei um eine sehr zukunftsträchtige Programmiersprache.

Auch der objektorientierte Ansatz zeigt, dass es sich bei Java um eine moderne Programmiersprache handelt. Diese Technik erlaubt nicht nur eine hohe Effizienz beim Programmieren. Darüber hinaus sind die Strukturen damit deutlich einfacher zu verstehen. Aus diesem Grund kommt die objektorientierte Programmierung heutzutage bei vielen Projekten zum Einsatz. Daher ist es sinnvoll, sich bereits frühzeitig in dieses Thema einzuarbeiten.

Sehr kontrovers wird das Thema diskutiert, ob eine strenge Syntax mit festen Regeln hilfreich für Programmieranfänger ist. Es gibt Sprachen wie beispielsweise Python, die hierbei keine strengen Regeln vorgeben. Das sorgt am Anfang für schnelle Fortschritte. Java ist hierbei hingegen sehr strikt. Das ist insbesondere beim Umgang mit verschiedenen Datentypen von großer Bedeutung. Auf der einen Seite mag es am Anfang etwas länger dauern, bis man diese Aspekte versteht und umsetzen kann. Später ist dies jedoch sehr hilfreich. Wenn man mit anderen Sprachen ohne strenge Typisierung beginnt, muss man sich bei anspruchsvolleren Anwendungen ebenfalls in dieses Thema einarbeiten. Zu diesem Zeitpunkt fällt das häufig jedoch schwerer, als wenn man sich bereits von Anfang an daran gewöhnt.

Schließlich gehört Java zur sogenannten C-Familie. Viele Programmiersprachen orientieren sich hinsichtlich ihrer Syntax und ihrer Befehle an C. Beispiele hierfür sind neben Java C++, C#, Perl, PHP und einige weitere Sprachen. Daher bestehen hierbei große Ähnlichkeiten. Wenn man später einmal andere Anwendungen programmieren will, für die die Verwendung einer anderen Programmiersprache sinnvoll ist, ist der Umstieg daher recht einfach. Aus diesem Grund ist es empfehlenswert, mit einer Sprache aus der C-Familie zu beginnen – beispielsweise mit Java.

Kapitel 2

Notwendige Programme für das Programmieren in Java

Bereits in der Einleitung wurde erwähnt, dass für die Erzeugung eines Java-Programms einige zusätzliche Software erforderlich ist. Dabei ist insbesondere das Java Development Kit (JDK) von großer Bedeutung. Ohne dieses kann man kein Java-Programm ausführen. Bevor man das erste Programm schreibt, ist es daher sinnvoll, diese Software zu installieren.

Das ist jedoch nicht die einzige Vorbereitungsmaßnahme, die wir treffen müssen. Darüber hinaus ist ein Programm erforderlich, mit dem wir den Quellcode verfassen. Des Weiteren verwenden wir eine integrierte Entwicklungsumgebung (IDE – Integrated Development Environment). Diese vereinfacht die Programmentwicklung und ist daher sehr hilfreich.

Die folgenden Abschnitte stellen daher vor, wie wir unseren Rechner vorbereiten müssen. Darin wird erklärt, wo die benötigten Programme erhältlich sind und was bei der Installation zu beachten ist. Dabei kommt ausschließlich Gratis-Software zum Einsatz, die wir ohne die Entrichtung von Lizenzgebühren verwenden dürfen. Daher fallen hierfür keine weiteren Kosten an.

2.1 Ein Texteditor für die Erstellung des Programmcodes

Wenn wir ein Programm in Java schreiben, fügen wir die entsprechenden Befehle in Textform ein. Das bedeutet, dass wir eine passende Software benötigen, die es uns erlaubt, diesen Text zu schreiben. Nun könnte man auf die Idee kommen, hierfür ein Textverarbeitungsprogramm wie Word zu verwenden. Dieses ist zum Programmieren jedoch vollkommen ungeeignet, da es neben dem eigentlichen Inhalt noch viele weitere Informationen zum Layout in den entsprechenden Dateien abspeichert. Daher kann der Compiler die entsprechenden Dateien nicht verarbeiten.

Für die Erstellung des Programmcodes benötigen wir einen sogenannten Texteditor. Dieser speichert lediglich den Text ab, den wir beim Programmieren eingeben. Außerdem erlaubt es die Software, ein passendes Dateiformat zu wählen. Auch das ist eine wichtige Voraussetzung dafür, dass wir das Programm später kompilieren können.

Ein einfacher Texteditor ist auf den meisten Betriebssystemen bereits vorinstalliert. Unter Windows ist dies beispielsweise der Microsoft Editor – auch bekannt unter der Bezeichnung Notepad. Dieser eignet sich zwar prinzipiell zum Programmieren. Aller-

dings ist er in seinem Funktionsumfang so stark eingeschränkt, dass es empfehlenswert ist, ein etwas hochwertigeres Programm zu verwenden. Leser, die für die Bearbeitung dieses Lehrbuchs einen Linux-Rechner verwenden, können diesen Abschnitt jedoch überspringen. Bei allen gängigen Distributionen ist bereits ein Texteditor enthalten, der für unsere Anwendungen vollkommen ausreicht – beispielsweise GEdit unter Ubuntu oder Kate unter KDE.

Eine der wesentlichen Vorteile eines höherwertigen Texteditors besteht darin, dass er den Code deutlich übersichtlicher gestaltet. Beispielsweise rückt er zusammengehörige Blöcke automatisch ein und gestaltet unterschiedliche Schlüsselbegriffe in verschiedenen Farben.

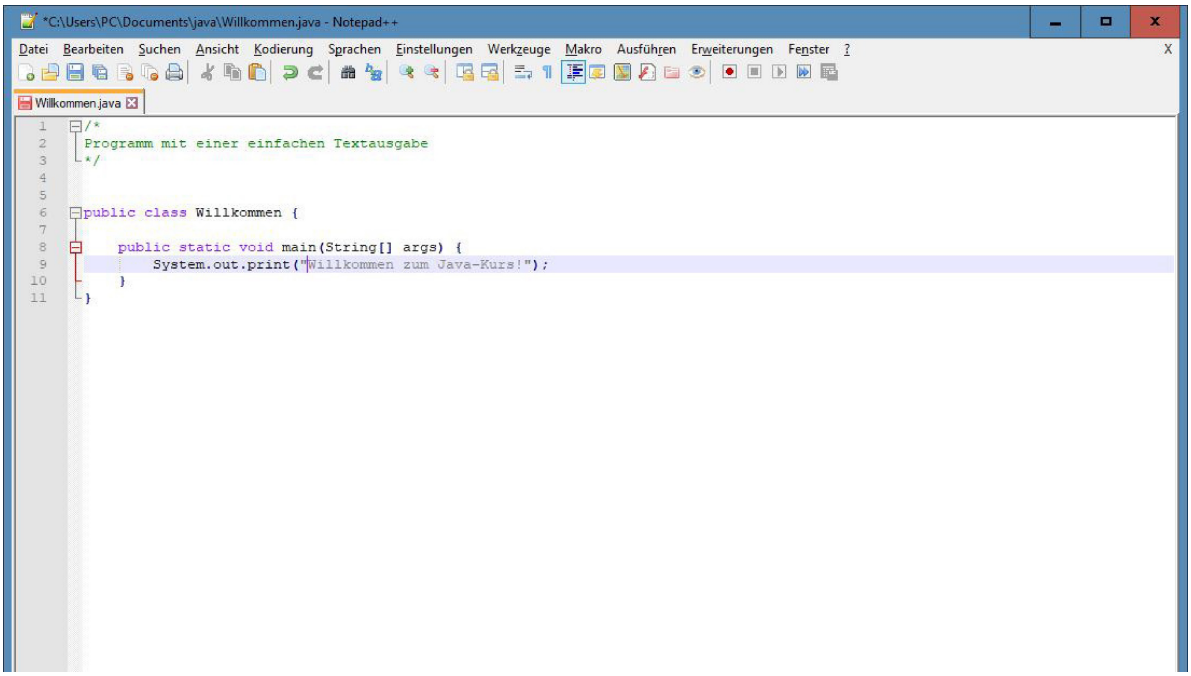

Abb. 2.1 Die Darstellung des Programmcodes in einem Texteditor

Bei längeren Programmen wirkt sich auch die seitliche Nummerierung der Zeilen sehr positiv aus. Das erleichtert es, den Überblick zu behalten. Außerdem lassen sich Programmblöcke, die man im Moment nicht bearbeiten will, einklappen. Das erhöht die Übersichtlichkeit deutlich.

Es gibt unzählige Angebote für praktische Texteditoren. Viele von ihnen sind kostenfrei erhältlich. Dennoch zeichnen sie sich durch einen hohen Funktionsumfang aus. In diesem Buch verwenden wir den Texteditor Notepad++. Hierbei handelt es sich um eine Open Source Software, sodass es möglich ist, sie ohne die Entrichtung einer Lizenzgebühr zu verwenden. Sie steht unter folgendem Link zum Download bereit: https://notepad-plus-plus.org/download/. Es ist jedoch auch möglich, selbst einen anderen Texteditor zu suchen und diesen zu installieren. Die Aufgaben in diesem Buch lassen sich damit ebenfalls bearbeiten.

2.2 Java Development Kit: wichtiges Werkzeug zum Programmieren

Im Einleitungskapitel zu diesem Buch wurde bereits angesprochen, dass es vor der Ausführung eines Java-Programms notwendig ist, dieses zu kompilieren. Dafür ist ein Compiler notwendig. Diese Software müssen wir daher ebenfalls auf unserem Rechner installieren. Sie ist im Java Development Kit enthalten.

Um das Entwicklungs-Kit herunterzuladen, besuchen wir folgende Seite:

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

Abb. 2.2 Die Downloadseite für JDK

Hier finden wir im oberen Bereich stets einen Link für den Download der aktuellen JDK-Version. Diesen klicken wir an. Daraufhin erscheint eine neue Seite. Hier müssen wir zum einen die Lizenzbedingungen akzeptieren und zum anderen eine passende Version für das verwendete Betriebssystem auswählen.

Wenn man die entsprechende Datei heruntergeladen hat, öffnet sich der Installationsassistent. Hierbei kann man stets die Standardeinstellungen übernehmen. Lediglich an der Stelle, an der der Assistent nach dem Installationsverzeichnis fragt, ist es wichtig, sich den Pfad zu merken – insbesondere wenn man hier ein anderes Verzeichnis als vorgeschlagen auswählt.

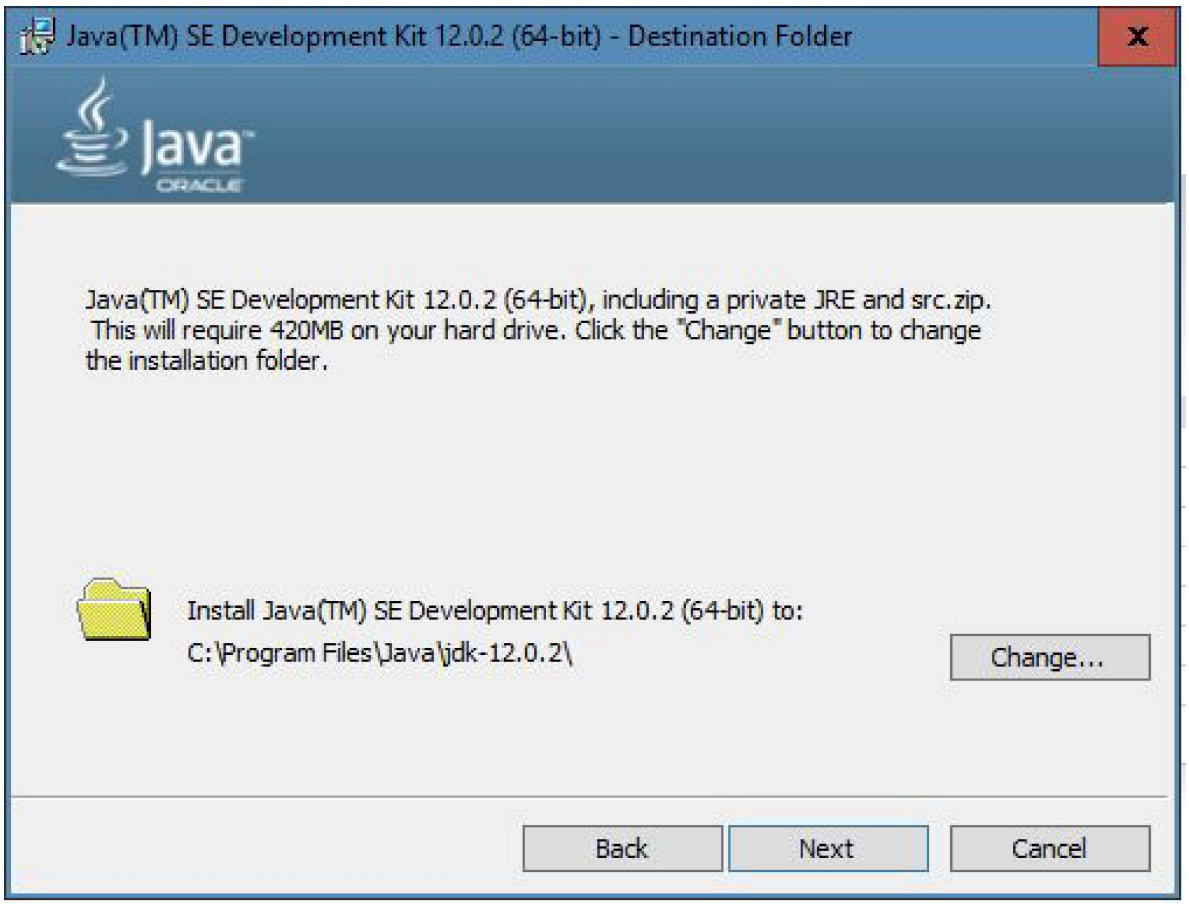

Abb. 2.3 Hier ist es wichtig, sich den Installationspfad zu merken

2.3 Vorbereitungsmaßnahmen für die Verwendung von JDK

Um zu überprüfen, ob JDK ordnungsgemäß installiert wurde, ist es sinnvoll, den Kommandozeileninterpreter zu öffnen. Dazu müssen wir im Windows-Startmenü den Ordner Windows-System und daraufhin den Begriff Eingabeaufforderung auswählen. Unter Linux ist die entsprechende Funktion unter dem Begriff Terminal verfügbar.

Um die Installation zu überprüfen, fragen wir die installierte JDK-Version ab. Das erfolgt mit folgendem Begriff:

```
javac -version
```

In der Regel erscheint nun jedoch eine Fehlermeldung, die besagt, dass der Begriff entweder falsch geschrieben oder nicht gefunden wurde.

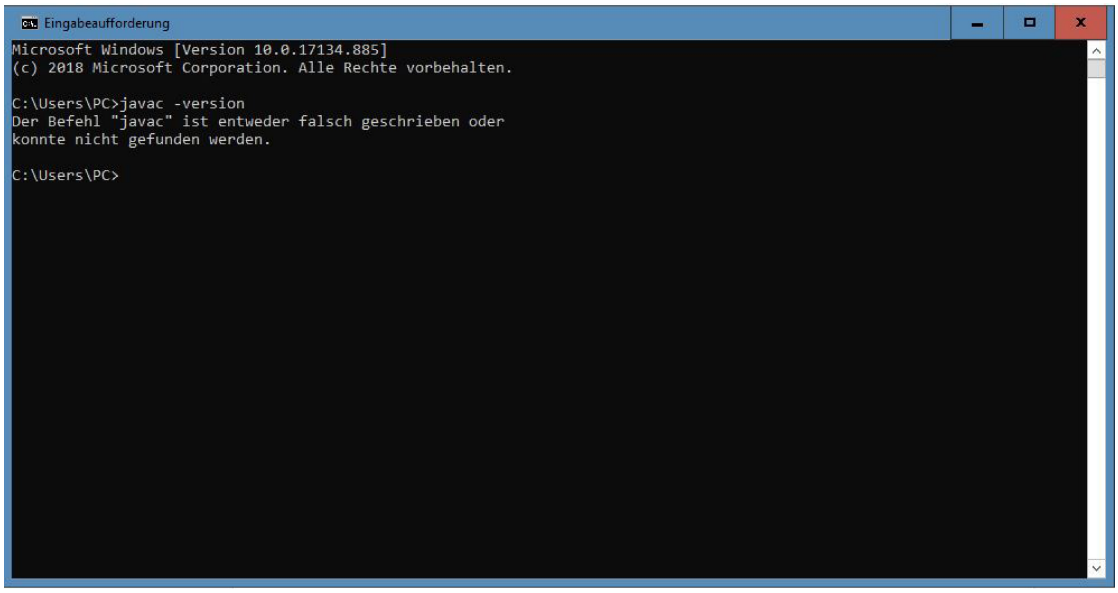

Abb. 2.4 Die Fehlermeldung bei der Abfrage der Version

Das muss jedoch noch nicht heißen, dass die Installation fehlgeschlagen ist. Um dies zu überprüfen, wechseln wir in das Verzeichnis, in dem wir JDK installiert haben. Dazu müssen wir zunächst in das Stammverzeichnis wechseln. Das erledigen wir, indem wir so oft cd.. eingeben und mit der Eingabetaste bestätigen, bis in der Zeile für die Eingabe nur noch der Ausdruck C:\> erscheint. Danach geben wir wieder das Kommando cd ein, gefolgt vom Pfadnamen für die Installation, den wir im vorherigen Abschnitt ausgewählt haben – allerdings ohne den vorangestellten Ausdruck C:\, der das Laufwerk vorgibt. Daran hängen wir den Ausdruck \bin an, um in das gewünschte Verzeichnis zu gelangen. Nun kann man den Befehl für die Versionsabfrage nochmals wiederholen. Jetzt sollte der Kommandozeileninterpreter die verwendete Version anzeigen:

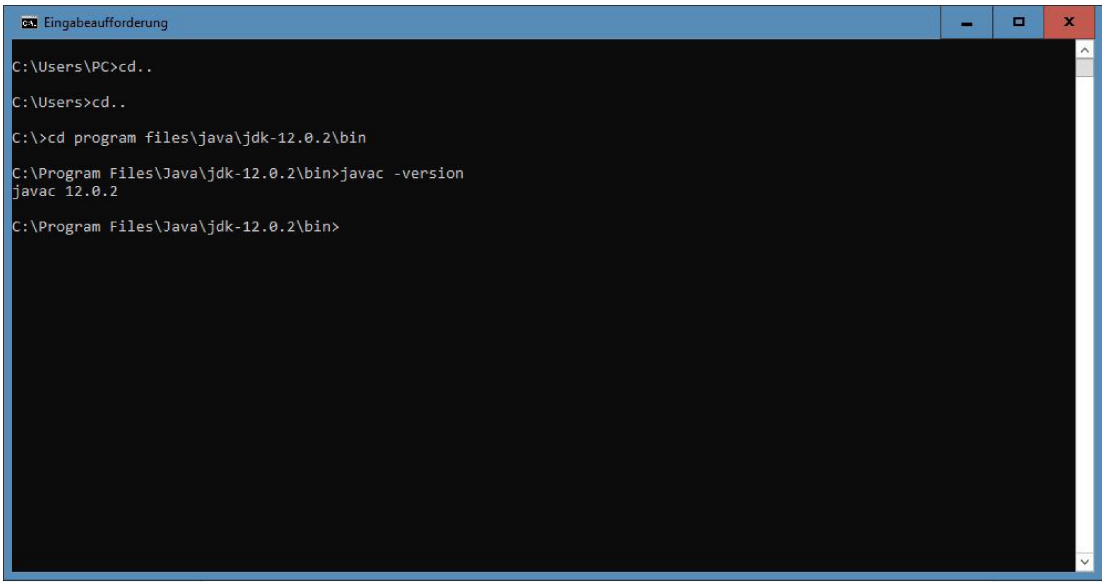

Abb. 2.5 Die erfolgreiche Abfrage der Version

Das zeigt, dass das JDK richtig installiert wurde. Wenn wir uns im richtigen Verzeichnis befinden, können wir das Programm nutzen. Daraus ergibt sich jedoch ein Problem. Wenn wir ein Programm schreiben, speichern wir dieses in der Regel nicht im Installationsverzeichnis vom JDK. Allerdings ist nur hier der Zugriff auf den Compiler möglich. Wenn wir alle Programme hier abspeichern, würde das die Übersichtlichkeit stark einschränken. Daher besteht der nächste Schritt darin, das JDK aus jedem beliebigen Verzeichnis verfügbar zu machen.

Zu diesem Zweck müssen wir unter Windows eine Umgebungsvariable hinzufügen. Hierfür gibt es mehrere Möglichkeiten. Eine Vorgehensweise besteht darin, in die Windows-Suchfunktion den Begriff „Erweiterte Systemeinstellungen“ einzugeben. Alternativ kann man auch das Zahnrad im Startmenü anklicken und danach die erweiterten Einstellungen auswählen. Daraufhin öffnet sich folgendes Fenster:

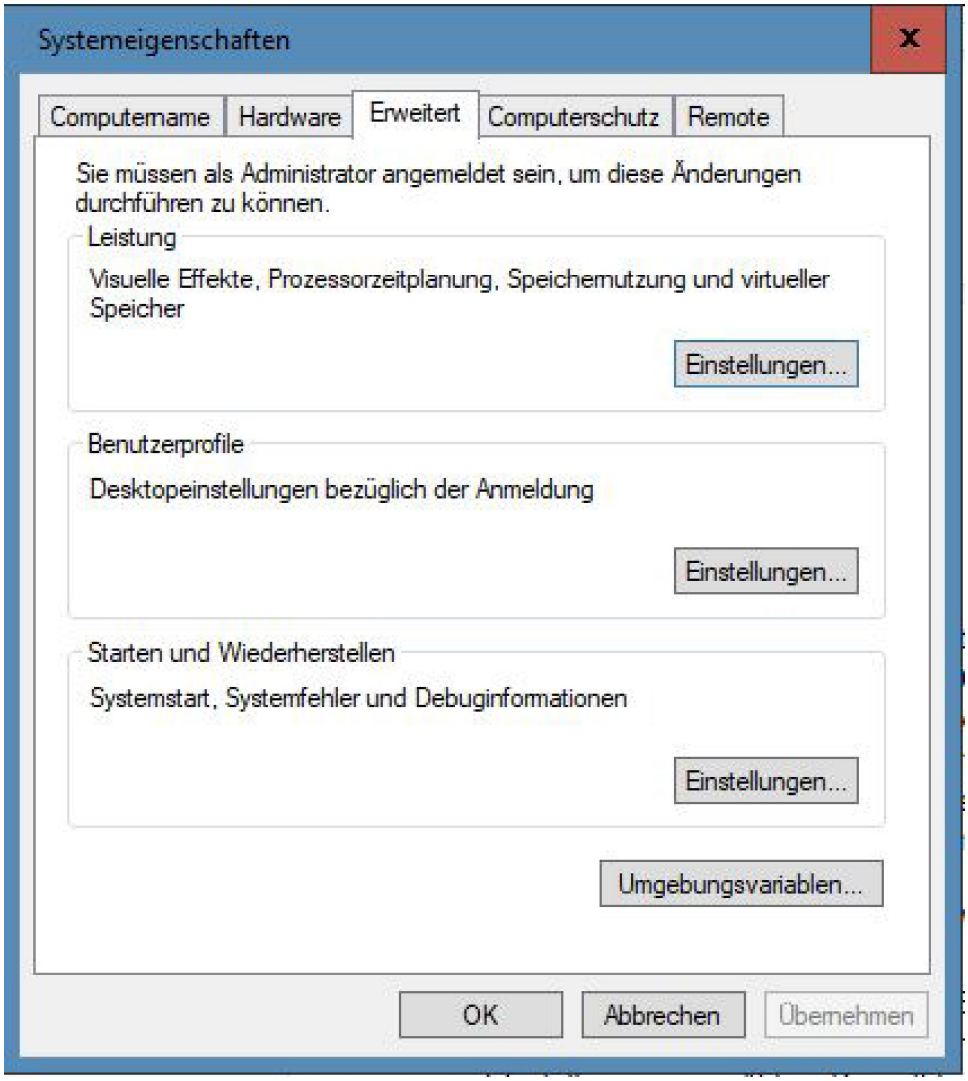

Abb. 2.6 Die erweiterten Systemeinstellungen

Hier klicken wir nun im unteren Bereich auf den Begriff „Umgebungsvariablen“. Daraufhin öffnet sich ein neues Fenster:

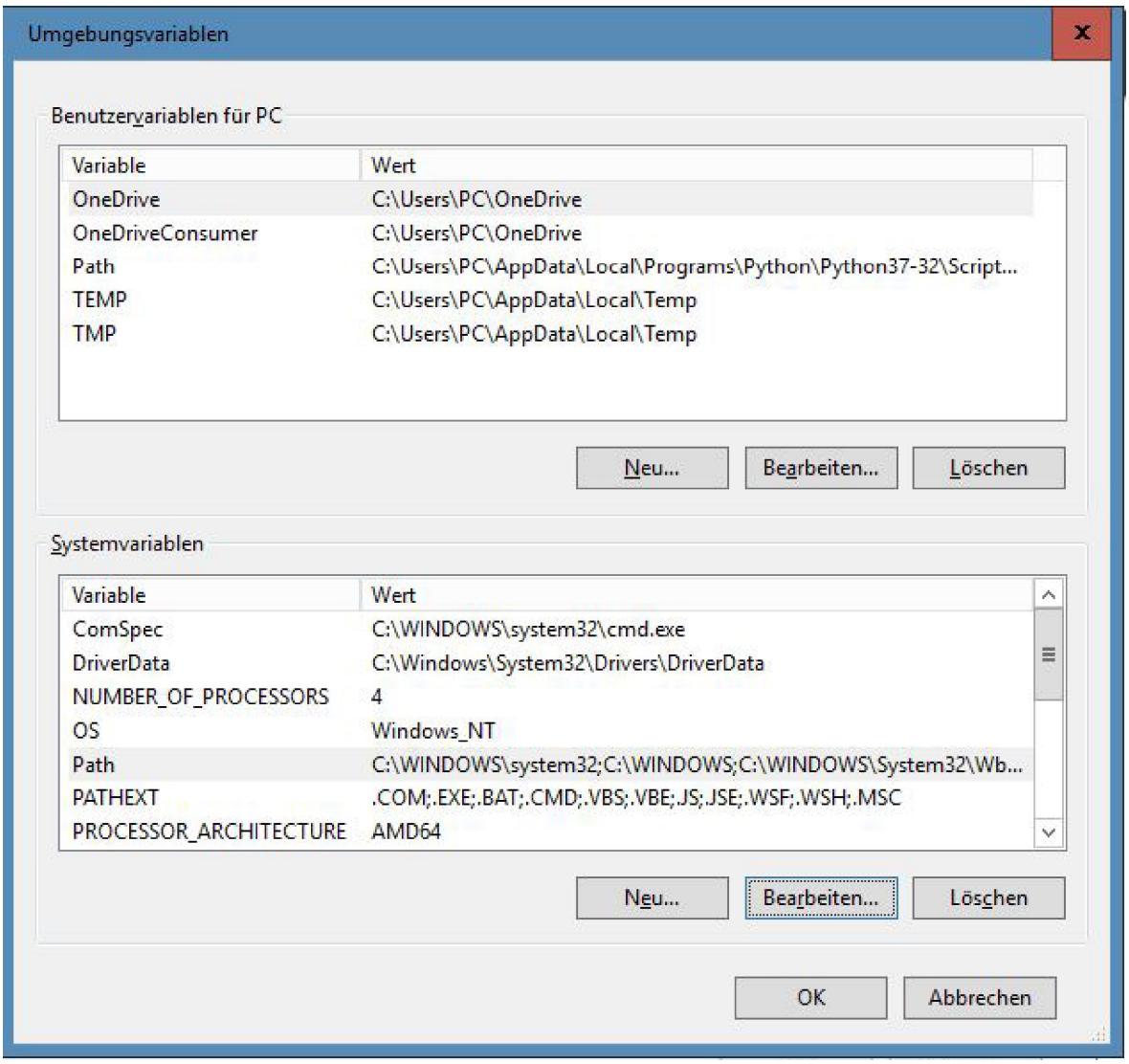

Abb. 2.7 Die vorhandenen Systemvariablen

Hier erscheinen nun zwei Listen. Wir benötigen lediglich den unteren Bereich mit der Aufschrift Systemvariablen. In der Regel ist hierbei bereits ein Eintrag mit der Bezeichnung Path vorhanden. Diesen wählen wir aus und klicken auf Bearbeiten (Sollte dieser Eintrag noch nicht vorhanden sein, müssen wir in diesem Fall auf „Neu“ klicken und eine Variable mit der Bezeichnung „Path“ erstellen). Daraufhin öffnet sich ein neues Fenster, in dem die bereits vorhandenen Pfadvariablen aufgeführt sind. Hier klicken wir nun auf „Neu“ und fügen den Pfadnamen der Installationsdatei mit dem Zusatz \bin hinzu. In unserem Beispiel sieht der Eintrag demnach so aus:

```
C:\program files\java\jdk-12.0.2\bin
```

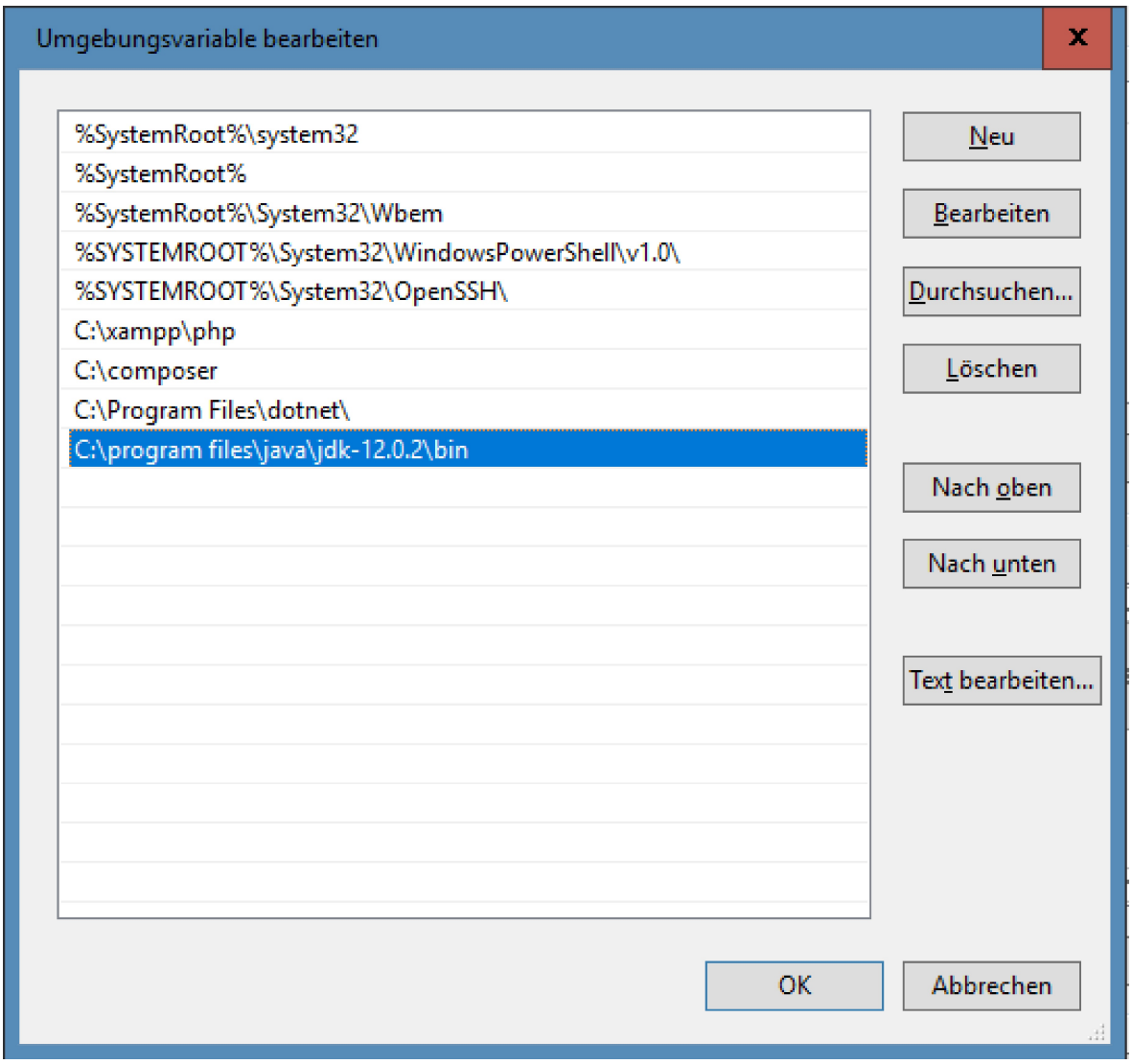

Abb. 2.8 Die neu hinzugefügte Umgebungsvariable

Nun muss man nur noch alle Fenster mit OK bestätigen. Um auszuprobieren, ob diese Anpassung erfolgreich verlief, müssen wir den Kommandozeileninterpreter schließen und daraufhin neu öffnen. Nun sollte es möglich sein die JDK-Version in jedem beliebigen Verzeichnis abzufragen:

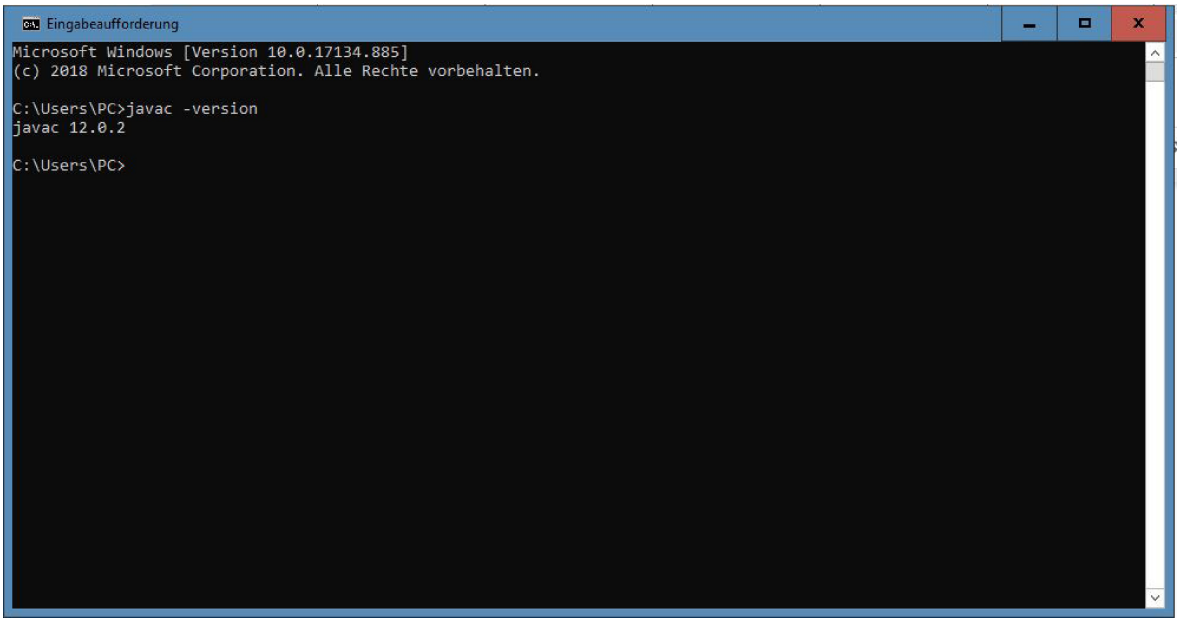

Abb. 2.9 Nun ist das JDK in jedem beliebigen Verzeichnis verfügbar

2.4 Die IDE installieren

Für die Gestaltung eines Java-Programms ist es möglich, dieses in einem Texteditor zu schreiben. Danach müssen wir zum Kommandozeileninterpreter wechseln und es kompilieren. Daraufhin können wir das Programm dort auch ausführen. Hierbei handelt es sich um die traditionelle Vorgehensweise. Die ersten Java-Programme, die wir im nächsten Kapitel gestalten werden, erstellen wir auf diese Weise. Allerdings ist mittlerweile die Verwendung einer IDE gebräuchlich. Diese übernimmt die Funktion des Texteditors, sodass wir hier den Programmcode erstellen können. Allerdings hat sie auch Zugriff auf den Compiler. Daher können wir unsere Programme direkt in der IDE kompilieren. Das erleichtert den Entwicklungsprozess erheblich. Außerdem bietet sie noch viele weitere hilfreiche Funktionen. Daher soll die traditionelle Vorgehensweise nur im folgenden Kapitel zum Einsatz kommen. Danach lernen wir die Verwendung der IDE kennen und nutzen diese für alle weiteren Programme.

Auch hinsichtlich der Auswahl einer passenden IDE gibt es sehr viele Auswahlmöglichkeiten. Die meisten dieser Programme sind ebenfalls gratis erhältlich. Es gibt zahlreiche Angebote, die das Programmieren mit Java unterstützen. Allerdings bestehen dabei große Unterschiede. Zahlreiche IDEs sind eigentlich auf eine andere Programmiersprache ausgerichtet. Die Unterstützung von Java wurde dabei nur nachträglich hinzugefügt. Das hat zur Folge, dass hierfür viele nützliche Funktionen nicht verfügbar sind.

Wenn wir uns auf das Programmieren mit Java konzentrieren möchten, ist es daher sinnvoll, eine IDE auszuwählen, die speziell für diese Programmiersprache entworfen wurde. Hierfür gibt es im Wesentlichen zwei verschiedene Angebote: Eclipse und Net-

Beans. Für dieses Lehrbuch kommen beide Alternativen infrage. Da es jedoch notwendig ist, eine Entscheidung zu treffen, soll die IDE NetBeans verwendet werden. Diese wurde für lange Zeit von Sun und später von Oracle entwickelt. Sie stammte daher vom gleichen Anbieter wie die Programmiersprache selbst. Das führte zu einer besonders engen Verbindung. Außerdem sind viele Java-Tutorials, die von Oracle stammen, auf die Verwendung von NetBeans ausgerichtet. Das kann sehr hilfreich sein. Mittlerweile hat Oracle dieses Projekt allerdings an Apache abgegeben.

Für den Download öffnen wir folgenden Link:

https://netbeans.apache.org/download/nb111/nb111.html

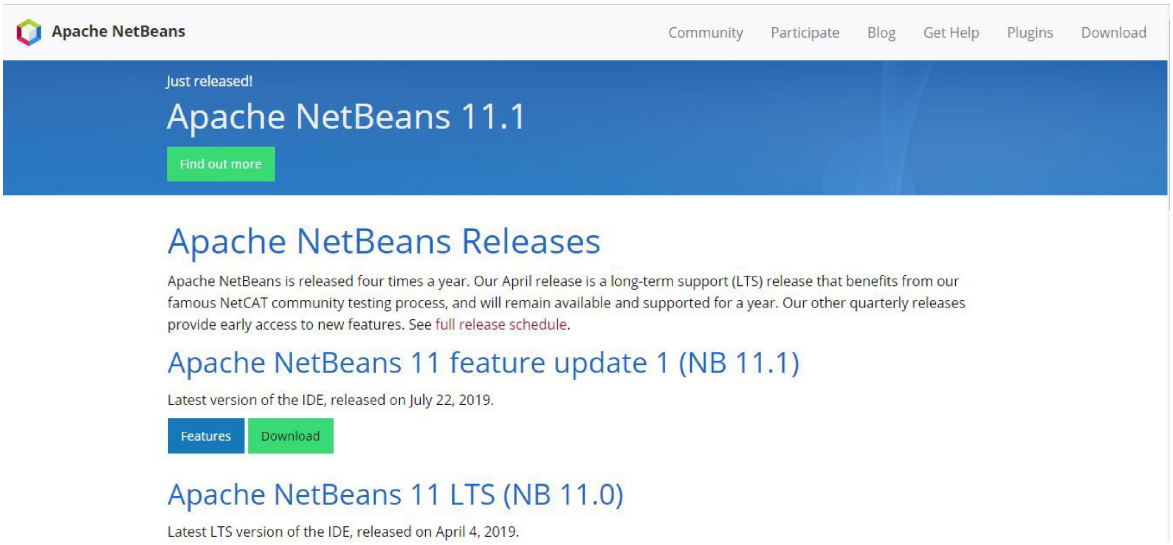

Abb. 2.10 Die Downloadseite für NetBeans

Wenn wir den Download-Button anklicken, werden wir zunächst zu einer weiteren Seite weitergeleitet, auf der wir eine passende Version für das verwendete Betriebssystem auswählen können. Für die Installation können wir einfach den Anweisungen des Assistenten folgen. Dieser sucht auch automatisch nach dem eben installierten JDK, um die Anwendung zu verknüpfen.

Kapitel 3

Die ersten Schritte mit Java

Nachdem alle Vorbereitungsmaßnahmen abgeschlossen sind, können wir nun damit beginnen, unser erstes Java-Programm zu schreiben. Damit der Einstieg so leicht wie möglich ist, soll dies nur eine sehr einfache Funktion haben: Es soll einen Text ausgeben.

Dazu ist selbstverständlich ein geeigneter Ausgabebefehl notwendig. Darüber hinaus müssen wir jedoch darauf achten, die vorgegebenen Strukturen einzuhalten. Jedes Java-Programm muss bestimmte Grundelemente enthalten, damit man es kompilieren kann. Dieses Kapitel stellt nicht nur vor, wie das Programm aufgebaut ist und welche Bedeutung die einzelnen Elemente haben. Darüber hinaus lernen wir, wie wir das Programm kompilieren und ausführen.

3.1 Hallo Welt – das erste eigene Programm schreiben

Einen Text auszugeben, ist wohl die einfachste Funktion, die ein Computerprogramm ausführen kann. Aus diesem Grund beginnen beinahe alle Lehrbücher für Programmier-Anfänger mit dieser Aufgabe. In vielen Beispielen lautet der Text, den das Programm ausgeben soll „Hallo Welt!“ – beziehungsweise in englischsprachigen Lehrbüchern „Hello World!“. Deshalb hat sich für dieses einfache Beispiel der Begriff Hallo-Welt-Programm eingebürgert.

Bei diesem Programm steht selbstverständlich der Ausgabebefehl im Mittelpunkt. Dieser sieht in Java so aus:

```
1  System.out.print("Hallo Welt!");
```

Der eigentliche Befehl für die Ausgabe des Texts lautet print. Allerdings kann dieser nicht alleine stehen. In Java sind alle Befehle in sogenannten Klassen angeordnet, die die einzelnen Funktionen zusammenfassen. Die Klasse, in der sich der `print`-Befehl befindet, trägt den Namen System. Dabei ist es wichtig, darauf zu achten, dass diese stets mit einem Großbuchstaben beginnen muss. Danach folgen ein Punkt und der Begriff `out`. Dieser sorgt für eine weitere Unterteilung. Erst danach kann man den eigentlichen Befehl anfügen.

Der Text, den wir ausgeben wollen, steht immer in einer Klammer nach dem `print`-Befehl. Das macht deutlich, dass sich der entsprechende Inhalt auf dieses Kommando bezieht. Wenn wir hier einen Text ausgeben möchten, müssen wir diesen stets in Anführungszeichen setzen. Nach dem Ausdruck folgt

ein Semikolon. In Java ist es notwendig, jeden Befehl mit diesem Zeichen abzuschließen.

Damit haben wir bereits den Befehl für die Ausgabe erstellt. Allerdings ist es nicht möglich, diesen alleine auszuführen. Es ist notwendig, ihn in verschiedene strukturelle Elemente einzubinden, über die jedes Java-Programm verfügen muss. Am Anfang stehen hierbei immer die Begriffe public und class. Der erste von ihnen sagt aus, dass der Zugang von jedem beliebigen Bereich aus möglich ist. Das ist eine wichtige Voraussetzung dafür, dass wir später mit dem Kommandozeileninterpreter auf das Programm zugreifen können, um es auszuführen.

Der zweite Ausdruck ist notwendig, um eine neue Klasse zu erzeugen. Dabei handelt es sich um das grundlegende Strukturelement der objektorientierten Programmierung. Was das genau bedeutet, wird später in diesem Buch noch erklärt. Bislang ist es lediglich notwendig zu wissen, dass jedes Java-Programm in einer Klasse stehen muss. Hierfür können wir einen beliebigen Namen auswählen. Zulässig sind dabei jedoch nur Buchstaben, Zahlen, der Unterstrich und das Dollarzeichen. Leerzeichen dürfen darin nicht enthalten sein und außerdem ist es verboten, die Bezeichnung mit einer Zahl zu beginnen. Des Weiteren ist es nicht erlaubt, reservierte Schlüsselbegriffe (wie beispielsweise `public`) zu verwenden. Es ist üblich, Klassennamen stets mit einem Großbuchstaben zu beginnen. Wenn wir unsere Klasse passend zur Programmfunktion `HalloWelt` nennen, ergibt sich daraus folgender Ausdruck:

```
1 public class HalloWelt {
```

Der Inhalt der Klasse steht stets in einer geschweiften Klammer. Danach folgt folgender Ausdruck:

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

Der Begriff main öffnet das Hauptprogramm, das stets den Einstiegspunkt für die Ausführung darstellt. Der Begriff `public` wurde bereits erklärt. Die anderen Begriffe sollen an dieser Stelle noch nicht genau erläutert werden – das geschieht im weiteren Verlauf des Buchs. Es ist nur wichtig, das Hauptprogramm stets mit dieser Zeile einzuleiten. Der Inhalt steht dann wieder in einer geschweiften Klammer. Damit ist das erste Programm bereits erstellt. Der komplette Code dafür sieht so aus:

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

Nun müssen wir das Programm nur noch abspeichern. Dabei ist es wichtig, darauf zu achten, dass der Dateiname genau dem Namen der Klasse entspricht, die das Hauptprogramm umschließt. Ist das nicht der Fall, lässt sich das Programm nicht kompilieren. Als Dateiendung wählen wir hierfür stets `.java`. Daraus folgt, dass wir die Datei unter dem Namen `HalloWelt.java` abspeichern müssen.

3.2 So lässt sich das Programm ausführen

Bislang existiert unser erstes Java-Programm nur in Textform. Das Ziel besteht jedoch stets darin, die Programme auszuführen, damit sie alle Aktionen, die wir darin vorgegeben haben, durchführen. Zu diesem Zweck müssen wir wieder den Kommandozeileninterpreter öffnen. Mit dem bereits bekannten Befehl **cd** wechseln wir daraufhin in das Verzeichnis, in dem wir das Programm abgespeichert haben. Der Pfad kann dabei von Anwender zu Anwender verschieden sein – je nachdem, für welche Verzeichnisstruktur er sich entschieden hat.

Nachdem wir uns im richtigen Verzeichnis befinden, müssen wir das Programm kompilieren. Dafür ist der Befehl javac erforderlich. Danach müssen wir den Namen unserer Datei inklusive ihrer Endung schreiben. Um das erste Programm zu kompilieren, verwenden wir daher folgenden Befehl:

```
javac HalloWelt.java
```

Nachdem wir dieses Kommando eingegeben haben, kommt es zunächst zu einer kleinen Wartezeit. Das zeigt bereits, dass es sich bei der Kompilierung um einen sehr rechenintensiven Prozess handelt – selbst bei diesem sehr einfachen Programm. Danach erscheint wieder die Zeile für die Eingabe weiterer Befehle. Es scheint daher auf den ersten Blick, als sei nichts passiert. Wenn wir jedoch den Ordner öffnen, in dem wir das Programm abgespeichert haben, erkennen wir, dass dieses Kommando doch eine Wirkung hatte. Hier ist nun eine neue Datei entstanden, die den Namen **HalloWelt.class** trägt. Dabei handelt es sich um das kompilierte Programm.

Dieses können wir im nächsten Schritt nun ausführen. Dafür müssen wir zunächst den Befehl **java** eingeben. Danach folgt wieder der Dateiname. In diesem Fall müssen wir diesen jedoch ohne eine Dateiendung nennen. Das entsprechende Kommando sieht dann so aus:

```
java HalloWelt
```

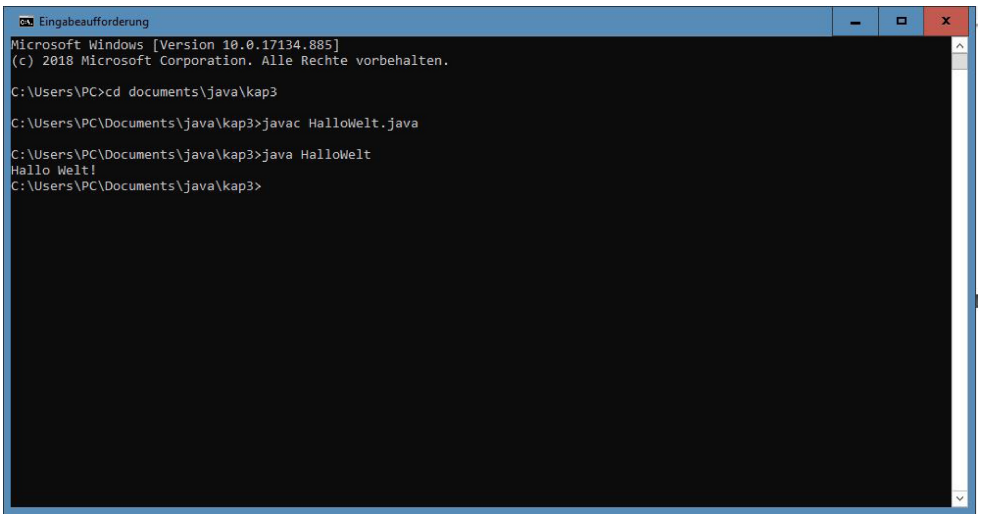

Abb. 3.1 Die Kompilierung und die Ausführung des Programms

Wenn wir diesen Befehl eingeben, erscheint im Kommandozeileninterpreter der Text, den wir in unser Programm geschrieben haben. Das zeigt, dass es perfekt funktioniert und seine Aufgabe erfüllt. Damit ist unser erstes Java-Programm bereits abgeschlossen.

3.3 Kommentare: weitere Informationen zum Programmcode hinzufügen

Der Code eines Computerprogramms ist nicht immer einfach zu verstehen. Wenn beispielsweise nach einiger Zeit ein Programmierer ein Programm überarbeitet, das er nicht selbst erstellt hat, fällt es häufig schwer, die Funktionsweise zu verstehen. Um dieses Problem zu verhindern, ist es möglich, sogenannte Kommentare einzufügen. Dabei handelt es sich um einfachen Text, den man für die Weitergabe wichtiger Informationen verwenden kann – beispielsweise um die Funktionsweise eines Programms zu erklären. Allerdings verfügt dieser Text über eine besondere Kennzeichnung. Diese macht deutlich, dass er nicht zum eigentlichen Code gehört. Daher wird er bei der Kompilierung nicht beachtet. Er ist lediglich im Quellcode zu erkennen.

Java kennt verschiedene Arten von Kommentaren. Für einzeilige Kommentare kommt ein doppelter Schrägstrich (`//`) zum Einsatz. Alles, was in dieser Zeile nach dieser Kennzeichnung steht, wird als Kommentar behandelt. Längere Kommentare werden mit einem Schrägstrich und einem anschließenden Sternsymbol (`/*`) geöffnet. Um sie zu beenden, verwendet man die gleichen Zeichen – allerdings in der umgekehrten Reihenfolge (`*/`).

Darüber hinaus gibt es in Java noch eine besondere Form des Kommentars – den Dokumentationskommentar. Dieser wird mit einem Schrägstrich und dem doppelten Sternsymbol (`/**`) geöffnet. Um ihn zu schließen, verwenden wir die gleiche Zeichenfolge wie bei gewöhnlichen Kommentaren. Wenn wir das Programm kompilieren und ausführen verhalten sich Dokumentationskommentare genau gleich wie gewöhnli-

che Kommentare. Allerdings bieten sie noch eine weitere Möglichkeit: Sie erlauben es, eine Dokumentation zu erstellen. Hierfür gibt es ein spezielles Programm, das den Namen javadoc trägt. Dieses liest den Programmcode aus und stellt dessen Struktur dar. Zu diesem Zweck erstellt es ein HTML-Dokument, das man mit einem beliebigen Webbrowser betrachten kann. Dabei gibt es alle Dokumentationskommentare an, sodass ersichtlich wird, welche Aufgabe die einzelnen Teile erfüllen. Wenn man diese Funktion nutzen will, ist es üblich, die entsprechenden Dokumentationskommentare vor Klassen, Attributen und Methoden anzubringen.

Wenn wir unser erstes Programm mit Kommentaren ausstatten, könnte es so aussehen:

```
1  /*
2  Dieses Programm ist das erste Beispiel in unserem Java-Lehrbuch.
3  Es soll die Ausgabe einer einfachen Textnachricht erklären.
4  */
5  /**
6  Die Klasse HalloWelt gibt eine Textnachricht aus.
7  */
8  public class HalloWelt {
9      public static void main(String[] args) {
10     //Hier steht unser Text.
11         System.out.print("Hallo Welt!");
12     }
13 }
```

Mit diesem Code können wir nun einmal das javadoc-Programm ausprobieren. Dieses ist bereits im JDK enthalten, sodass wir es nicht installieren müssen. Um es zu verwenden geben wir im Kommandozeileninterpreter einfach den Begriff **javadoc** gefolgt vom Dateinamen ein. Hierfür müssen wir uns selbstverständlich im richtigen Verzeichnis befinden. Wenn wir daraufhin das Verzeichnis öffnen, sehen wir, dass eine Vielzahl an neuen Dateien entstanden ist. Wenn wir nun die Datei HalloWelt.html anklicken, öffnet sich folgende Seite:

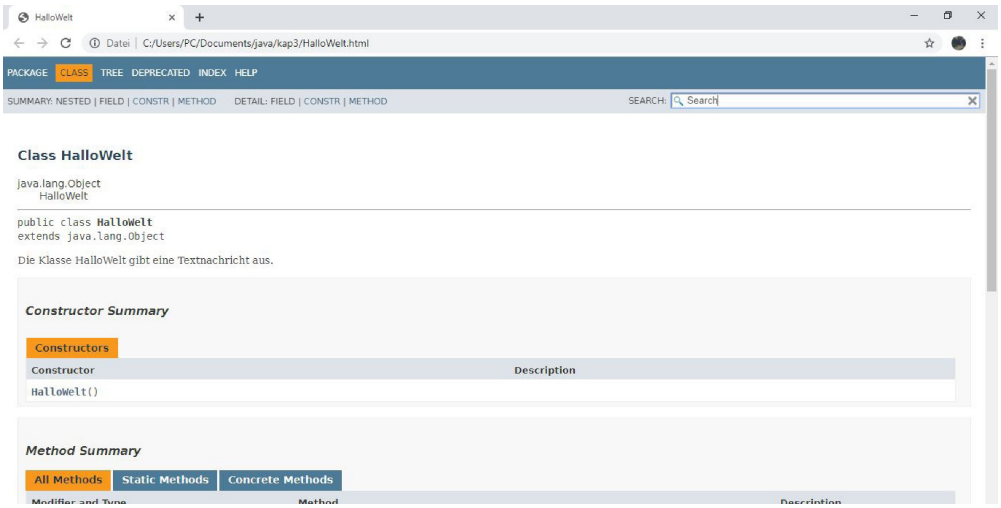

Abb. 3.2 Die automatisch generierte Dokumentation zu unserer Datei

Hier erkennen wir, dass die HTML-Datei nicht nur die Struktur unseres Programms abbildet. Darüber hinaus erscheint hier unser javadoc-Kommentar.

Kommentare können zwar hilfreich sein, doch nimmt ihre Verwendung immer weiter ab. Der Grund dafür besteht darin, dass es mittlerweile als guter Stil angesehen wird, den Programmcode so zu schreiben, dass er selbsterklärend ist. Das macht Kommentare überflüssig. Dennoch ist es wichtig, diese Funktionsweise kurz vorzustellen. In der Informatik muss man immer wieder mit Code arbeiten, den andere Programmierer erstellt haben. Dabei trifft man häufig auf Kommentare. In diesen Fällen ist es wichtig, zu wissen, dass diese keine Auswirkung auf die Funktionen des Programms haben.

Während Kommentare im fertigen Programm mittlerweile nicht mehr üblich sind, kommen sie während der Erstellung nach wie vor häufig zum Einsatz. Beispielsweise ist es möglich, hier einen kurzen Vermerk anzubringen, wenn man eine bestimmte Aufgabe erst später erledigen will. Außerdem ist es möglich, dass ein Bereich noch Fehler oder Verweise zu noch nicht erstellten Methoden enthält. Das macht es unmöglich, das Programm zu kompilieren. Wenn man den entsprechenden Code jedoch als Kommentar kennzeichnet, ist die Ausführung möglich. Diese Technik wird als Auskommentieren bezeichnet.

3.4 Übungsaufgabe: Erste Java-Programme selbst gestalten

Am Ende der meisten Kapitel befindet sich eine kleine Übungsaufgabe. Diese gibt dem Leser die Möglichkeit, den Stoff zu wiederholen und selbst anzuwenden. Das vertieft den Lerneffekt deutlich und führt dazu, dass sich die Inhalte besser einprägen. Daher ist es empfehlenswert, diese Aufgaben zu bearbeiten.

Ihr Schwerpunkt liegt stets auf den Techniken, die im jeweiligen Kapitel vermittelt wurden. Doch werden auch alle zuvor erlernten Befehle als bekannt vorausgesetzt. In seltenen Fällen sind für die Bearbeitung auch neue Kenntnisse erforderlich. Diese werden dann stets in der entsprechenden Aufgabe erklärt.

Das Ziel dieser Übungsaufgaben besteht darin, dass der Leser diese selbstständig löst. Selbst wenn das nicht auf Anhieb gelingen sollte, ist es sinnvoll, es noch ein weiteres Mal zu probieren. Zum Abschluss wird dennoch stets eine Musterlösung angegeben. Diese dient in erster Linie dazu, sie mit der eigenen Lösung zu vergleichen. Nur wenn man selbst überhaupt nicht weiterkommt, ist es sinnvoll, hier nachzuschauen.

In der Informatik gibt es in der Regel nicht nur einen einzigen Lösungsweg. Häufig bestehen viele verschiedene Alternativen, um die gewünschte Funktionsweise zu erreichen. Das heißt, dass auch die hier angegebene Musterlösung nicht die einzige Möglichkeit darstellen muss. Wenn das Programm, das Sie entwickelt haben, die

Anforderungen der Aufgabe erfüllt, handelt es sich dabei um eine korrekte Lösung – selbst wenn der Code von der hier angegebenen Musterlösung abweicht.

Für dieses Kapitel müssen Sie nur zwei kleine Aufgaben lösen, diese wiederholen die Inhalte, die soeben vermittelt wurden:

1. Erstellen Sie ein Programm, das den Anwender zu unserem Java-Kurs begrüßt. Kompilieren Sie den Code und führen Sie ihn aus.
2. Fügen Sie in das Programm zwei Kommentare ein. Verwenden Sie hierfür unterschiedliche Auszeichnungsarten.

Lösungen:

1.

```
public class Aufgabe1 {
    public static void main(String[] args) {
        System.out.print("Willkommen zum Java-Kurs!");
    }
}
```

2.

```
/*
Übungsaufgabe für die Verwendung von Kommentaren
*/
public class Aufgabe2 {
    public static void main(String[] args) {
    //Der print-Befehl gibt unsere Nachricht aus.
        System.out.print("Willkommen zum Java-Kurs!");
    }
}
```

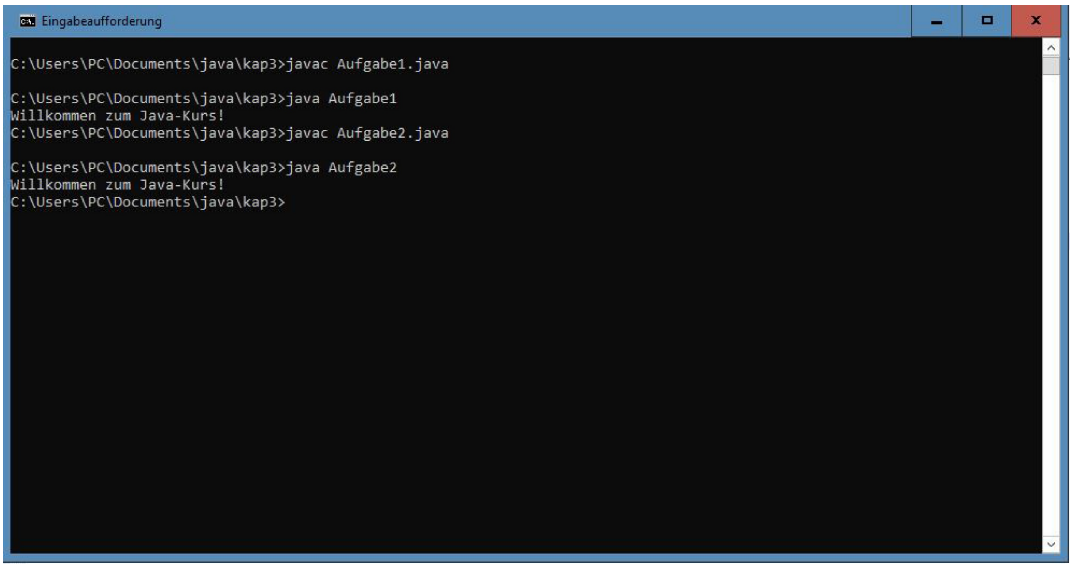

Abb. 3.3 Die Ausgabe ist bei beiden Programmen identisch

Alle Programmcodes und Schaltpläne aus diesem Buch stehen kostenfrei zum Download bereit. Dadurch müssen Sie Code nicht abtippen.

Außerdem erhalten Sie die eBook Ausgabe zum Buch im PDF Format kostenlos auf unserer Website: