Hans-Peter Habelitz

Programmieren lernen mit Java

Keine Vorkenntnisse erforderlich

- Vom ersten Programm bis zur fertigen Anwendung
- Mit vielen Beispielen und Übungsaufgaben
- Inkl. Objektorientierung, GUI-Entwicklung, Datenbanken u. v. m.

Alle Codebeispiele und Musterlösungen zum Download

Rheinwerk Computing

Kapitel 1
Einführung

Wer die Zukunft erforschen will, muss die Vergangenheit kennen.
– Chinesische Weisheit

Die Programmiersprache Java wird an vielen Schulen und Hochschulen als Basis für den Einstieg in die Programmierung verwendet. Beim Erlernen dieser erfolgreichen Programmiersprache möchte dieses Buch Sie unterstützen. Besonderer Wert wurde bei der Erstellung darauf gelegt, dass keine Programmierkenntnisse vorausgesetzt werden, sodass Sie das Programmieren von Grund auf lernen können. Bei der Auswahl der Entwicklungsumgebung wurde darauf geachtet, dass diese eine komfortable Arbeit ermöglicht und dass sie frei und für alle gängigen Betriebssysteme (also Windows, Linux und macOS) gleichermaßen verfügbar ist. *Eclipse* ist für den Einstieg nicht ganz einfach; Sie werden mit diesem Buch aber den Umgang damit lernen und leicht damit arbeiten können.

Dieses Buch führt Sie schrittweise in die Programmierung mit *Java* ein. Sie lernen alle wichtigen Sprachstrukturen anhand von Anwendungsbeispielen kennen, damit Sie schon bald Ihre eigenen Programme entwickeln können. An die meisten Kapitel schließen sich Übungsaufgaben an. Durch deren Bearbeitung wird der Umgang mit den neuen Sprachelementen und Strukturen eingeübt und gefestigt. Musterlösungen zu den Übungsaufgaben finden Sie als Quellcodedateien im Downloadmaterial oder im Anhang. Verweise auf die betreffenden Fundstellen sind im Text angegeben. Neben den Musterlösungen erhalten Sie als Downloadmaterial auch die Versionen des *JDK* (*Java Development Kit*) und der Entwicklungsumgebung Eclipse für die Betriebssysteme Windows, Linux und macOS. Da in einem einführenden Buch nicht alle Sprachdetails bis in die letzten Einzelheiten behandelt werden können, stellt das Buch »Java ist auch eine Insel« vom Rheinwerk Verlag eine sehr gute Ergänzung dar. Weitere Hinweise zu ergänzender Literatur, die zum Teil kostenlos im Internet zur Verfügung steht, sind im Literaturverzeichnis am Ende des Buches zusammengestellt.

1.1 Was bedeutet Programmierung?

Bevor Sie mit dem Programmieren loslegen können, brauchen Sie ein Grundverständnis dafür, was Programmierung überhaupt ist und wie sie »funktioniert«. Das sollen Ihnen die nächsten Abschnitte näherbringen.

1.1.1 Von den Anfängen bis heute

Das erste Computerprogramm, das jemals erstellt wurde, wird einer Frau zugeschrieben. Die britische Mathematikerin *Ada Lovelace* (1815–1852) entwickelte einen schriftlichen Plan, wie man mithilfe der mechanischen Rechenmaschine von *Charles Babbage* (1791–1871) Bernoulli-Zahlen berechnen kann. Das ist umso erstaunlicher, weil zu diesem Zeitpunkt lediglich Pläne für diese Rechenmaschine vorlagen. Diese mechanische Rechenmaschine (*Analytical Engine*), die zu Lebzeiten ihres Erfinders nie gebaut wurde, gilt als Vorläufer der heutigen Computer, und der Plan von Ada Lovelace wird als das erste Computerprogramm angesehen.

Seit der Erfindung der ersten mechanischen Rechenmaschinen bis zu den heutigen elektronischen Computersystemen haben sich viele weitreichende Veränderungen eingestellt. Das gilt sowohl für die Hardware als auch für die Arbeitsmittel, die zur Programmierung verwendet werden. Allerdings haben die grundlegenden Zusammenhänge bis heute ihre Gültigkeit bewahrt. Die frühen Programmierwerkzeuge orientierten sich noch sehr an der Hardwarestruktur und machten es notwendig, dass der Programmierer die Bausteine des Prozessors explizit kannte und ansprechen konnte. Zum Addieren der beiden Zahlen 12 und 38 können Sie in Java einfach die Anweisung

```
x = 12 + 38;
```

verwenden. Diese Schreibweise ist nicht neu, denn wir verwenden sie auch in der Mathematik, wenn die Variable `x` die Summe der beiden Zahlen 12 und 38 annehmen soll. In anderen aktuellen Programmiersprachen wie C/C++ sieht eine Addition genauso aus, und sie gilt unabhängig vom verwendeten Rechner bzw. dem darin verbauten Prozessor. In einer älteren hardwarenahen Sprache wie Assembler mussten Sie dafür etwa folgende Anweisungsfolge verwenden:

```
mov eax, 12
add eax, 38
```

Zuerst wurde die Zahl 12 in ein Prozessorregister mit dem Namen `eax` geschrieben (`mov` steht für das englische *move*), um in einem zweiten Schritt den Registerinhalt um den

Wert 38 zu erhöhen. Der Programmierer musste z. B. wissen, wie die Register des Prozessors heißen und mit welchen Registern eine Addition ausgeführt werden kann. Dass unterschiedliche Prozessoren auch unterschiedliche Registerbezeichnungen verwenden können, hat das Programmieren zusätzlich erschwert.

Die frühen Computersysteme waren so einfach aufgebaut, dass dies auch noch zu leisten war. Moderne Computersysteme sind heute so komplex und entwickeln sich so schnell weiter, dass es kaum noch möglich ist, die Prozessordetails zu kennen. Glücklicherweise haben sich in dem gleichen Maße, in dem die Komplexität der Systeme zugenommen hat, auch die Programmierwerkzeuge weiterentwickelt. Zu diesen gehören Editoren, die schon beim Schreiben von Programmanweisungen auf mögliche Fehler aufmerksam machen und dabei helfen, den Programmtext übersichtlich zu formatieren. Auch Übersetzungsprogramme, die die Programmdateien so aufbereiten, dass sie auf verschiedenen Rechnern mit unterschiedlichen Prozessoren ausgeführt werden können, gehören dazu. Die Programmiersprachen haben sich der menschlichen Sprache angenähert und können wesentlich leichter als die frühen, sehr hardwarenahen Sprachen erlernt werden.

1.1.2 Wozu überhaupt programmieren?

Die Programmierung, d. h. die Erstellung eines Computerprogramms, besteht darin, die Lösungsschritte für eine Problemstellung so zu formulieren, dass sie von einem Computersystem ausgeführt werden können. Das bedeutet, dass dem Programmierer die notwendigen Lösungsschritte bekannt sein müssen. Entweder muss er sich den Lösungsweg selbst erarbeiten oder dieser wird ihm zur Verfügung gestellt. Beim Programmieren wird dieser allgemein formulierte Lösungsweg in eine Programmiersprache übertragen, die vom Computersystem weiterverarbeitet werden kann.

Da die Programmierung einen zeitaufwendigen Prozess darstellt, muss die Frage beantwortet werden, wann es sich lohnt, diese Zeit zu investieren. Die Übertragung einer Aufgabenstellung auf ein Computersystem ist dann sinnvoll, wenn dieses System seine speziellen Fähigkeiten auch ausspielen kann. Diese Fähigkeiten sind vor allem:

- die hohe Verarbeitungsgeschwindigkeit
- die zuverlässige Wiederholbarkeit

Die hohe Verarbeitungsgeschwindigkeit kann nur genutzt werden, wenn die zeitaufwendige Programmerstellung nicht ins Gewicht fällt. Das ist immer dann der Fall, wenn das Programm häufig verwendet wird und oft seinen Geschwindigkeitsvorteil ausspielen kann. Gleiches gilt für die hohe Zuverlässigkeit. Im Gegensatz zum Menschen zeigt ein Computersystem bei der Ausführung sich ständig wiederholender Anweisungen

keinerlei Ermüdungserscheinungen. Konzentrationsfehler wegen Übermüdung sind ihm vollkommen fremd.

Die Arbeitsschritte zur Lösung einer Problemstellung werden allgemein auch als *Algorithmus* bezeichnet. Dieser Begriff wurde ursprünglich für die Beschreibung von Lösungswegen in der Mathematik verwendet und später auf die Informatik, die Wissenschaft, der die Programmierung zuzuordnen ist, übertragen. Jedem Computerprogramm liegt ein Algorithmus zugrunde. Deshalb liefert die Definition des Begriffs entscheidende Hinweise für die Beantwortung der Frage, ob eine Problemstellung mit einem Computerprogramm gelöst werden kann.

Ein Algorithmus muss die folgenden Anforderungen erfüllen:

- Er muss aus Arbeitsschritten bestehen, die zur Lösung einer Problemstellung führen.
- Er muss in einem endlichen Text vollständig beschreibbar sein.
- Jeder Schritt muss zu einem eindeutigen Zwischenergebnis führen.
- Er muss für gleiche Eingabewerte immer zum gleichen Ergebnis führen.
- Das Verfahren muss zum richtigen Ergebnis führen.
- Das Verfahren muss allgemeingültig sein, d. h., es muss auf alle zulässigen Daten anwendbar sein.

Die letzte Eigenschaft macht deutlich, dass ein Algorithmus der Lösung eines allgemeinen Problems dienen muss. Ein Programm, das nur die Zahlen 3 und 5 addieren kann, ergibt keinen Sinn – es muss in der Lage sein, zwei beliebige Zahlen zu addieren. Das bedingt aber, dass dem Programm mitgeteilt werden muss, welche beiden Zahlen addiert werden sollen. Dieses einfache Beispiel zeigt, dass die Lösung allgemeiner Probleme den Dialog zwischen Anwender und Programm notwendig macht.

Häufig erfordert der Entwurf eines Algorithmus als Vorstufe zur Programmierung Kreativität und Einfallsreichtum. Er wird oft als der schwierigste Teil im Prozess der Programmentwicklung bezeichnet.

1.1.3 Hilfsmittel für den Programmentwurf

Der Entwurf eines Algorithmus kann unabhängig von der zu verwendenden Programmiersprache erfolgen. Alle prozeduralen Programmiersprachen stellen die gleichen Sprachstrukturen zur Verfügung. Deshalb liegt es nahe, allgemeingültige Hilfsmittel zur Entwicklung von Algorithmen zu entwickeln und einzusetzen.

Computerprogramme bestehen sehr schnell aus umfangreichen Textdateien, die dann entsprechend unübersichtlich werden. Gerade in der Planungsphase ist es wichtig, den

Überblick zu behalten und eine Grobstruktur des fertigen Programms herauszuarbeiten, die in weiteren Phasen der Entwicklung verfeinert wird. Zur übersichtlichen Darstellung von Programmstrukturen eignen sich grafische Symbole. Für die Programmierung werden hierfür *Programmablaufpläne* (DIN 66001) oder *Nassi-Shneiderman-Struktogramme* (DIN 66261) verwendet. Die Gegenüberstellung in Abbildung 1.1 zeigt die in beiden Darstellungsformen verwendeten Symbole.

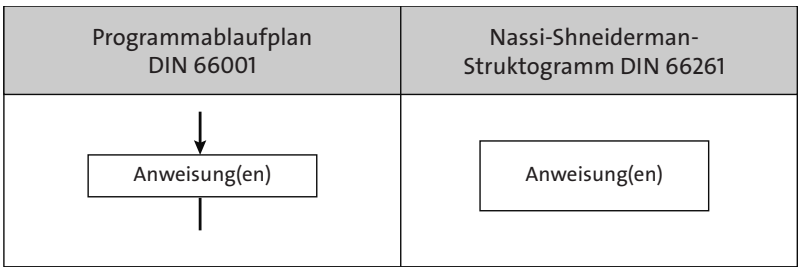


Abbildung 1.1 Einzelanweisung bzw. Anweisungsblock

In beiden Formen wird das Rechtecksymbol zur Darstellung einer einzelnen Anweisung oder eines zusammengehörenden Anweisungsblocks verwendet. Damit kann die Darstellung einer Programmlogik sehr detailliert und nahe am späteren Programmtext, aber auch sehr komprimiert dargestellt werden. In den Programmsymbolen können frei formulierte Aufgabenbeschreibungen stehen.

Mit einer *Anweisungsfolge* kann die Reihenfolge der Abarbeitung von Anweisungen verdeutlicht werden (siehe Abbildung 1.2). Im Programmablaufplan (PAP) wird die Reihenfolge der Abarbeitung zusätzlich durch Pfeile verdeutlicht. Die Abarbeitung in der Struktogrammdarstellung erfolgt grundsätzlich von oben nach unten.

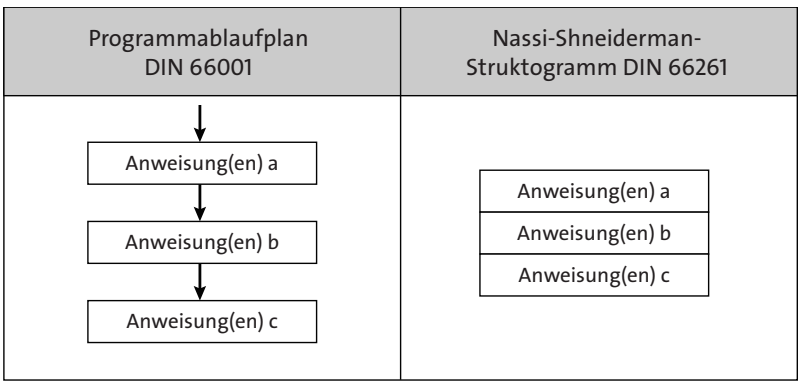


Abbildung 1.2 Anweisungsfolge

Lediglich drei Grundstrukturen werden benötigt, um einen Algorithmus zu beschreiben:

- Anweisungsfolge (Sequenz)
- Auswahlstruktur (Selektion)
- Wiederholungsstruktur (Iteration)

Zur Vereinfachung stellen Programmiersprachen unterschiedliche Varianten von Auswahl- und Wiederholungsstrukturen zur Verfügung.

Die einfachste Auswahlstruktur ist die *bedingte Verzweigung* (siehe Abbildung 1.3). Hierbei stehen zwei Alternativen zur Auswahl, die an eine Bedingung geknüpft sind. Ist die Bedingung erfüllt, wird die Anweisung bzw. werden die Anweisungen *a* ausgeführt, ansonsten die Anweisung bzw. die Anweisungen *b*.

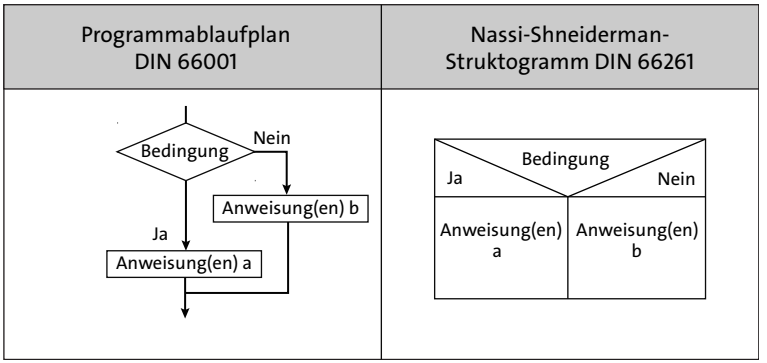


Abbildung 1.3 Bedingte Verzweigung

Wie diese Grundstrukturen in der Programmiersprache Java realisiert werden, erfahren Sie in Kapitel 3, »Kontrollstrukturen«. Im folgenden Abschnitt wird der gesamte Prozess der Anwendungsentwicklung anhand eines einfachen Beispiels erläutert.

1.1.4 Von der Idee zum Programm

Am Anfang des Entwicklungsprozesses steht immer eine Idee oder eine Vorstellung davon, was ein Programm leisten soll. Stellen Sie sich vor, Sie möchten ein kleines Computerspiel programmieren. Das Computerprogramm soll eine zufällig bestimmte ganze Zahl im Bereich von 1 bis 100 vorgeben, die vom Anwender erraten werden soll. Der Anwender soll beliebig viele Versuche haben, um die gesuchte Zahl zu erraten. Wurde die gesuchte Zahl erraten, soll das Programm die Anzahl der benötigten Versuche mitteilen.

Diese kurze Beschreibung steht für die *Produktdefinition*. Im professionellen Bereich spricht man vom *Lastenheft*, das möglichst präzise beschreibt, was das fertige Programm leisten soll. Für uns interessanter ist aber vielleicht eine Abwandlung des Lastenhefts, die als *User-Story* bezeichnet wird. Sie hat ihren Ursprung in der agilen Softwareentwicklung, beschränkt sich bei der Beschreibung der Anforderungen auf die Sicht des Nutzers und beantwortet möglichst kurz und allgemeinverständlich hauptsächlich die Fragen, *wer was warum* möchte. Die Produktdefinition, ganz gleich in welcher Form, stellt den Ausgangspunkt für den Entwicklungsprozess dar. Die Produktdefinition kann unter der Mitarbeit des Programmentwicklers entstehen oder auch vorgegeben sein.

In Abbildung 1.4 ist dargestellt, dass sich nach dem klassischen Modell der Programmentwicklung an die Produktdefinition vier weitere Phasen anschließen. Sie werden in der Regel wiederholt durchlaufen oder es gibt Rückgriffe. Unabhängig davon schließt sich an die Definition eines Programms eine Entwurfsphase an, bevor der erste Programmcode entsteht. Das Ergebnis der Entwurfsphase kann z. B. Programmablaufpläne, GUI-Skizzen (GUI steht für *Graphical User Interface* und zeigt, wie die Programmfenster aussehen, die sich dem Nutzer zeigen) oder Diagramme mit Komponenten des Programms enthalten.

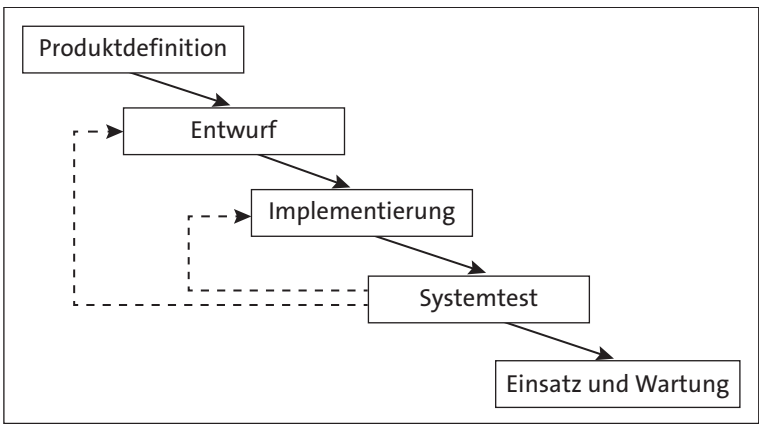


Abbildung 1.4 Phasen der Programmentwicklung

Die beiden in Abbildung 1.5 und Abbildung 1.6 gezeigten Darstellungsformen stehen gleichwertig nebeneinander. Während die Struktogrammdarstellung kompakter ist, ist der Programmablaufplan für Neueinsteiger etwas einfacher zu durchschauen.

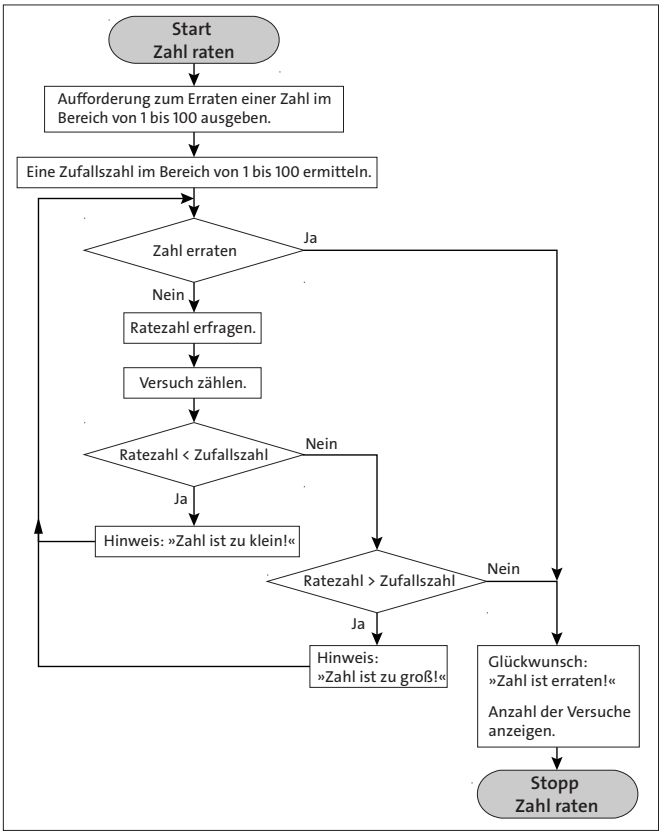


Abbildung 1.5 Programmentwurf als Programmablaufplan

Zahl raten		
Aufforderung zum Erraten einer Zahl im Bereich von 1 bis 100 ausgeben.		
Eine Zufallszahl im Bereich von 1 bis 100 ermitteln.		
Solange die Zahl nicht erraten ist, wiederhole:		
Ratezahl erfragen.		
Versuch zählen.		
Ratezahl < Zufallszahl		
Ja	Nein	
Hinweis: »Zahl ist zu klein!«	Ratezahl > Zufallszahl	
	Ja	Nein
	Hinweis: »Zahl ist zu groß!«	Glückwunsch: »Zahl ist erraten!« Anzahl der Versuche anzeigen.

Abbildung 1.6 Programmentwurf als Nassi-Shneiderman-Struktogramm

Unter *Implementierung* versteht man die Realisierung des Entwurfs in einer konkreten Programmiersprache. Listing 1.1 zeigt das Ergebnis der Implementierung in der Programmiersprache Java.

```
/*
 * Zahlraten
 * @version 1.0
 * @date 2024-06-06
 * @author Hans-Peter Habelitz
 */

package ratespiel;

import java.util.Random;
import javax.swing.JOptionPane;

public class Zahlraten {
   private int zufallszahl;
   private int ratezahl;

   Zahlraten() {
       ratezahl = -1;
   }

   void setZufallszahl(int zahl) {
      zufallszahl = zahl;
   }

   int getZufallszahl() {
      return zufallszahl;
   }

   void setRatezahl(int zahl) {
      ratezahl = zahl;
   }

   int getRatezahl() {
      return ratezahl;
   }
```

```
   public static void main(String[] args) {
      var spiel = new Zahlraten();
      var geraten = false;
      var versuchzaehler = 0;
      JOptionPane.showMessageDialog(null,
                  "Erraten Sie eine ganze Zahl aus dem Bereich von 1 bis 100!");
      var randomGenerator = new Random();
      spiel.setZufallszahl(randomGenerator.nextInt(101));
      while (!geraten) {
         spiel.setRatezahl(Integer.parseInt(
                 JOptionPane.showInputDialog("Welche Zahl wird gesucht?")));
         versuchzaehler++;
         if (spiel.getRatezahl() < spiel.getZufallszahl()) {
            JOptionPane.showMessageDialog(null, "Ihre Zahl ist zu klein!");
         } else {
            if (spiel.getRatezahl() > spiel.getZufallszahl()) {
               JOptionPane.showMessageDialog(null, "Ihre Zahl ist zu groß!");
            } else {
                geraten = true;
                JOptionPane.showMessageDialog(null,
            "Glückwunsch! Sie haben die Zahl mit "
            + versuchzaehler + " Versuchen erraten!");
            }
         }
      }
   }
}
```

Listing 1.1 »Zahl raten« als Java-Programm

Das Programm kann jetzt übersetzt und auf seine korrekte Funktion hin überprüft werden (*Systemtest*). Zeigen sich bei den Tests noch Fehler oder Unzulänglichkeiten, kann eine Überarbeitung des Entwurfs oder der Implementierung notwendig werden. Als letzte Phase schließt sich der Einsatz auf dem Zielsystem und die *Wartung* an. Zur Wartung gehören auch Erweiterungen und Anpassungen an veränderte Bedingungen im Umfeld.

Bereits dieses Beispiel zeigt, dass die grafischen Darstellungen wesentlich übersichtlicher ausfallen können als der in der Programmiersprache erstellte Programmcode. Die verwendeten Wiederholungs- und Auswahlstrukturen fallen deutlich ins Auge. Für Sie

als Anfänger ist es deshalb häufig hilfreich, dass Sie die Hilfsmittel kennen und nutzen, die zur Übersichtlichkeit des Programmcodes beitragen. Mit zunehmender Übung werden diese Hilfsmittel an Bedeutung verlieren und vielleicht auch nicht mehr genutzt, da man als Programmierer übersichtlich strukturierten Programmcode gut überblicken kann. Voraussetzung dafür ist jedoch, dass man sich an die Richtlinien zur übersichtlichen Gestaltung von Programmcode hält. Es handelt sich dabei um Richtlinien für die Schreibweise unterschiedlicher Programmkomponenten und um Einrückungen im Programmtext, die verdeutlichen, welche Codeteile zusammengehören. Diese Richtlinien werden Sie in Kapitel 2, »Grundbausteine eines Java-Programms«, kennenlernen.

Im folgenden Abschnitt möchte ich Ihnen einen Überblick über die verschiedenen Programmiersprachen geben. Sie sollen erfahren, welche unterschiedlichen Sprachen es gibt und worin sich diese unterscheiden. So können Sie die Programmiersprache Java in einem größeren Zusammenhang sehen.

1.1.5 Arten von Programmiersprachen

Programmiersprachen können nach folgenden Kriterien eingeteilt werden:

- Maschinennähe
- Anwendungsbereich
- Programmiermodell

Schauen wir uns im Folgenden diese drei Kriterien näher an.

Maschinennähe

Anwendungsprogramme werden erstellt, um von einer Maschine ausgeführt werden zu können. Mit »Maschine« ist hier jegliches Gerät gemeint, das Software enthält. Neben einem Computer kann es sich genauso gut um einen DVD-Player oder um ein Haushaltsgerät handeln. Damit ein Programm von einem Gerät ausgeführt werden kann, muss es in einer Sprache vorliegen, die von der Maschine verstanden wird. Die digitale Maschinensprache ist von unserer menschlichen Sprache sehr weit entfernt. Entwickeln Menschen Programme für Maschinen, ist entsprechend eine sehr große Distanz zu überbrücken. Unmittelbar in Maschinensprache zu programmieren würde bedeuten, dass man ausschließlich Zahlenkolonnen schreiben müsste, die sich aus Nullen und Einsen zusammensetzen (siehe Abbildung 1.7). Diese Art der Programmierung hat sich schon sehr früh als nicht praktikabel erwiesen und wurde von der maschinennahen Programmierung abgelöst.

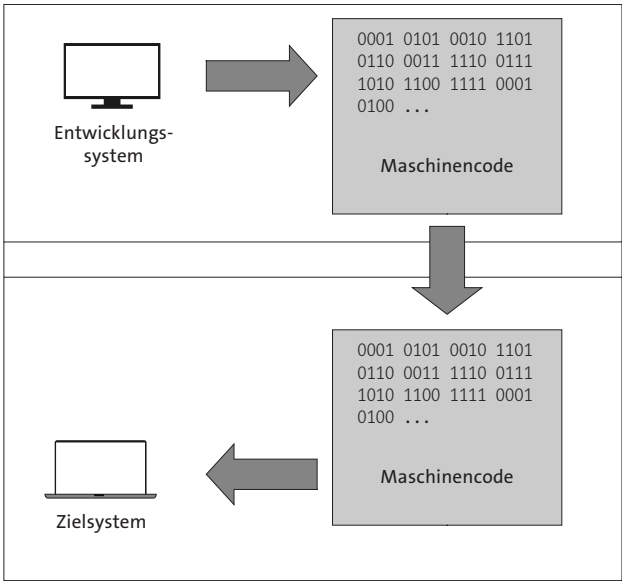


Abbildung 1.7 In Maschinensprache programmieren

Sie können sich das in etwa so vorstellen, dass für jede Zahlenkolonne ein Kürzel verwendet wird (z. B. `add` für »addiere«), das die Bedeutung der Zahlenkolonne verdeutlicht. Ein dafür erforderliches Übersetzungsprogramm hat es damit relativ leicht, jeden einzelnen Befehl in seine Entsprechung als Maschinenbefehl zu übersetzen. Etwas verwirrend ist die Namensgebung für das Übersetzungsprogramm, das den maschinennahen Code in Maschinencode überträgt. Es heißt ebenso wie die maschinennahe Sprache selbst *Assembler*.

Maschinennahe Programmiersprachen haben den Vorteil, dass sie alle Möglichkeiten eines *Prozessors* ausnutzen und für den Programmierer trotzdem noch einigermaßen lesbar sind. Der Prozessor ist das Herzstück des Computers. Er ist für die Programmausführung und damit auch für die Geschwindigkeit der Programmausführung verantwortlich. Jeder Prozessor verfügt über einen bestimmten Befehlssatz. Beim Übersetzen eines Programms in Maschinensprache werden die in der Programmiersprache erstellten Programmanweisungen in Befehle dieses Befehlssatzes übersetzt (siehe Abbildung 1.8).

Würde man unmittelbar in Maschinensprache programmieren, wäre keine Übersetzung erforderlich. Da sich maschinennahe Programmiersprachen sehr stark am Befehlssatz des Prozessors orientieren, lassen sich Programme optimal auf die Maschine abstimmen, die das Programm ausführen soll. Andererseits erfordern sie vom Programmierer sehr

viel Spezialwissen und schränken ein erstelltes Programm sehr stark auf eine bestimmte Maschine ein.

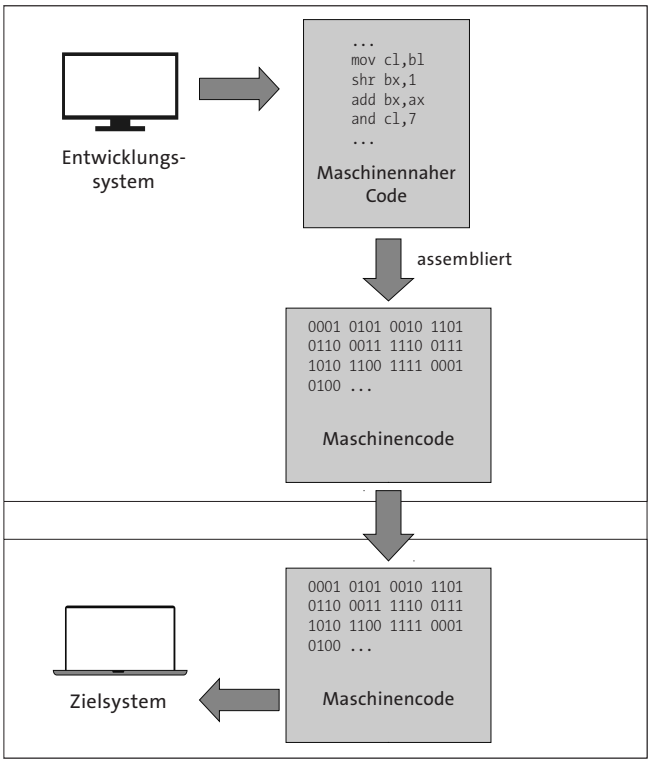


Abbildung 1.8 Maschinennahe Programmierung

Computer verwenden jedoch unterschiedliche Prozessoren mit sehr verschiedenen Befehlssätzen. Ein maschinennahes Programm kann, ebenso wie ein Maschinenprogramm, nur auf einem Prozessortyp laufen, weil es unmittelbar auf den Befehlssatz abgestimmt ist. Damit das gleiche Programm auch auf einem anderen Prozessor ausführbar wird, muss es unter Verwendung des Befehlssatzes dieses Prozessors neu erstellt werden. Entsprechend verursachen maschinennahe Sprachen sehr großen Aufwand, wenn es um die Pflege und *Portierung* der Programme geht. Unter Portierung versteht man das Übertragen auf ein anderes System.

Höhere Programmiersprachen, zu denen auch Java zählt, bilden den Gegenpol zu maschinennahen Sprachen. Sie setzen aufwendigere Übersetzungsprogramme ein. Der Programmierer erstellt das Programm in einem sogenannten *Quellcode*, der vor der Ausführung in den Maschinencode des Prozessortyps bzw. des darauf aufsetzenden

Betriebssystems übersetzt wird (siehe Abbildung 1.9). Zur Portierung von Programmen auf andere Prozessoren bzw. Betriebssysteme müssen lediglich Übersetzungsprogramme für diese Umgebungen zur Verfügung gestellt werden. Der Quellcode des eigentlichen Programms kann unverändert weiterverwendet werden.

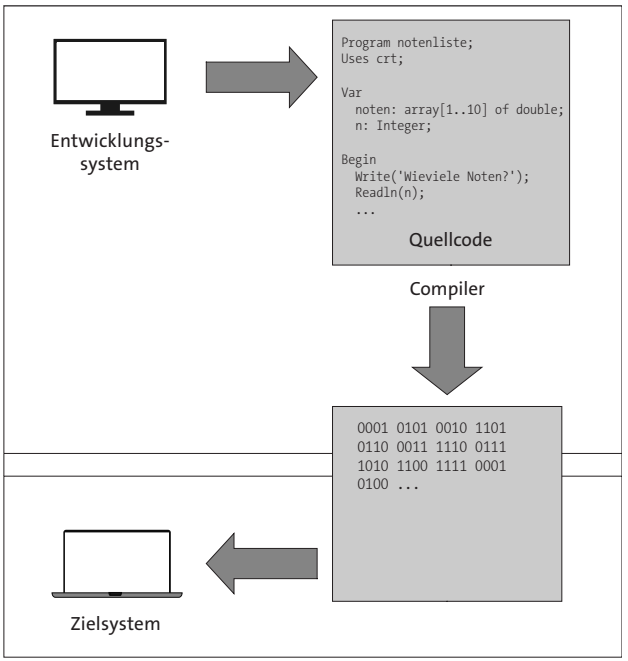


Abbildung 1.9 Hochsprachenprogrammierung mit Compiler

Übersetzungsprogramme, die den Quellcode komplett übersetzen und daraus eine eigenständige ausführbare Datei erstellen, nennt man *Compiler*. Unter dem Betriebssystem Windows sind von Compilern erzeugte Dateien an der Dateiendung *.exe* zu erkennen.

Übersetzungsprogramme, die den Quellcode bei der Ausführung nutzen sowie Anweisung für Anweisung übersetzen und an das Betriebssystem zur Ausführung weiterleiten, nennt man *Interpreter* (siehe Abbildung 1.10).

Kompilierte Programme haben Vorteile in der Ausführungsgeschwindigkeit, da der Code komplett übersetzt vorliegt und für die Übersetzung keine Zeit aufgewendet werden muss. Der Programmierer kann seinen Quellcode vor unrechtmäßigen Kopien schützen, indem er seine Quellcodedateien nicht weitergibt. Für jede Plattform muss eine eigene Maschinencodedatei erstellt werden. Das kompilierte Programm benötigt auf dem Zielsystem keine weiteren Hilfsmittel.

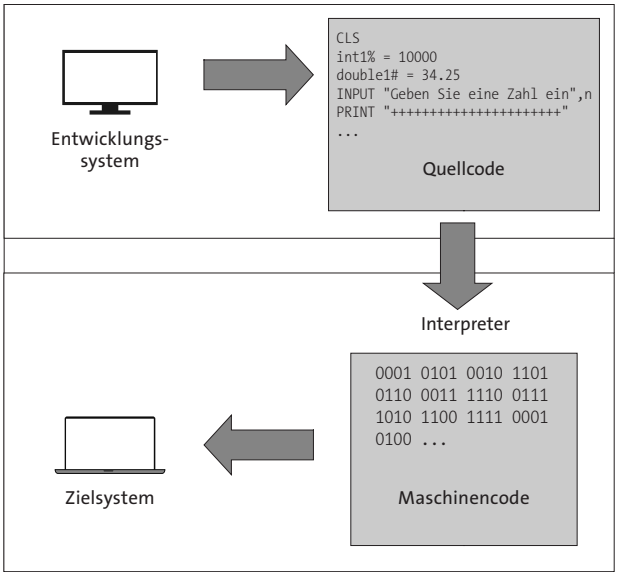


Abbildung 1.10 Hochsprachenprogrammierung mit Interpreter

Interpretierte Programme verwenden bei der Ausführung immer die Quellcodedatei. Jede Anweisung wird aus der Quellcodedatei ausgelesen und unmittelbar vor der Ausführung übersetzt. Das erfordert jeweils Zeit und verlangsamt den Programmablauf. Wer programmiert, muss bei der Weitergabe des Programms immer den eigenen Quellcode offenlegen. Auf der ausführenden Maschine muss außer dem Quellcode auch der passende Interpreter installiert sein, da er für die Übersetzung benötigt wird. Dennoch sind interpretierte Programme für den Entwickler vorteilhaft. So muss beispielsweise bei der Fehlersuche nicht vor jedem Testlauf das komplette Programm übersetzt werden. Auch wenn der Übersetzungsvorgang Fehler hervorbringt, kann das Programm zumindest bis zur fehlerhaften Stelle abgearbeitet werden und zeigt dem Programmierer Teilerfolge an. Außerdem kann das Programm »step by step«, d. h. Anweisung für Anweisung, abgearbeitet werden, und der Entwickler kann leicht nachvollziehen, welche Schritte jeweils durchgeführt werden.

Beide Sprachtypen haben ihre Berechtigung, da sie je nach Anwendungsfall ihre Stärken ausspielen können.

Anwendungsbereich

Programmiersprachen unterscheiden sich nicht nur technisch, sondern auch nach dem Anwendungsbereich, für den sie bestimmt sind oder für den sie sich besonders gut eignen. Generell kann zwischen Programmiersprachen unterschieden werden, die von

vornherein für einen bestimmten Zweck entwickelt wurden, und solchen, die als universelle Programmiersprachen entstanden sind. Auch die Entwicklung einer Programmiersprache entspricht dem Prozess einer Programmentwicklung. Eine Programmiersprache kann dem Programmierer viele Freiheiten lassen, aber genau diese können dazu führen, dass sehr schnell Fehler auftreten, die erst in der Testphase erkennbar werden und deren Ursachen nur sehr schwer aufzuspüren sind. Andererseits kann eine Programmiersprache die Möglichkeiten des Programmierers einschränken, um ihn zu zwingen, gut strukturiert zu programmieren, damit schon bei der Erstellung des Programms Fehler vermieden werden und eine Softwarepflege über einen längeren Zeitraum ohne großen Aufwand möglich ist.

Die Programmiersprachen *Pascal* und *Modula* wurden ursprünglich mit dem Ziel entwickelt, wohlstrukturierte Sprachen zur Verfügung zu stellen, die das Erlernen der Programmiergrundlagen erleichtern sollten. Inzwischen wurde Pascal um die Objektorientierung erweitert und steht unter dem Namen *Delphi* mit einer umfassenden Bibliothek visueller Komponenten zur Erstellung grafischer Benutzeroberflächen in direkter Konkurrenz zu *Visual C++* bzw. *C#*.

Fortran ist eine Programmiersprache, die sich besonders gut für naturwissenschaftlich-technische Anwendungen eignet. Mittlerweile ist diese Sprache allerdings kaum noch bekannt. An ihrer Stelle wird inzwischen *Python* für mathematische und naturwissenschaftliche Fragestellungen genutzt. Diese Sprache zeichnet sich dadurch aus, dass sie sehr gut lesbar und deshalb auch bei Anfängern sehr beliebt ist. Sprachen wie *JavaScript* und *PHP* können ihre Stärken ausspielen, wenn es um die Programmierung für das *World Wide Web* geht. Mit ihnen lassen sich hervorragend Skripte erstellen, die im Webbrowser bzw. auf einem Webserver ausgeführt werden und ihre Ergebnisse über Webseiten zur Verfügung stellen.

Eine der universellsten Programmiersprachen ist heute C bzw. mit den Erweiterungen C++ und C#. Sie gibt dem Programmierer sehr viele Freiheiten, die aber schnell zu den oben beschriebenen Problemen führen können. C# ist sehr gut für die Spiele-Entwicklung und zur Erstellung plattformübergreifender Anwendungen geeignet. C# und Java weisen übrigens viele Ähnlichkeiten auf, denn die Möglichkeit der plattformübergreifenden Programmerstellung ist auch eine große Stärke von Java.

Die Programmiersprache *Swift* wurde von *Apple* entwickelt. Sie greift Ideen von unterschiedlichen anderen Programmiersprachen auf und soll neben *Objective-C* eine zusätzliche Möglichkeit zur Programmentwicklung für die Plattformen *iOS* und *macOS* bieten.

Kotlin ist eine plattformübergreifende Programmiersprache. Die Plattformunabhängigkeit wird dadurch erreicht, dass Kotlin in Bytecode für die *Java Virtual Machine* (*JVM*)

übersetzt wird. Sie kann darüber hinaus aber auch in *JavaScript*-Quellcode oder in Maschinencode übersetzt werden.

Bei der Programmiersprache *Go* handelt es sich um eine Entwicklung von Google-Mitarbeitern; Go soll insbesondere die Softwareentwicklung für moderne Computersysteme im Hinblick auf *skalierbare Netzwerkdienste*, *Cloud-* und *Cluster-Computing* unterstützen.

Programmiermodell

Das Programmiermodell hat sich im Laufe der Jahre mehrfach geändert. Zuerst waren Programmiersprachen nach dem *imperativen Ansatz* gestaltet. Der Programmcode war aus linearen Listen von Anweisungen mit bedingten Anweisungen und Sprungbefehlen aufgebaut. Beispiele für diese Sprachen sind *Cobol* und *Fortran*. Demgegenüber sehen prozedurale Sprachen eine Trennung von Daten und Programmcode vor und bauen ein Programm aus Funktionen und Prozeduren auf, um den Gesamtcode zu strukturieren. Sprachen wie *Pascal* und *Modula* sind typische Vertreter dieser Programmiersprachen. Die genannten sind jedoch nicht die einzigen Programmiermodelle. Daneben existieren noch die weniger verbreiteten funktionalen (z. B. *F#* oder *Lisp*) und logischen Programmiersprachen (z. B. *Prolog*).

Die meisten modernen Programmiersprachen unterstützen die Objektorientierung. Bei diesem Programmiermodell bilden Objekte die Grundlage. Ziel ist es, eine Analogie zur realen Welt zu bilden, die ebenfalls aus Objekten besteht. Objekte können durch ihre Eigenschaften und Fähigkeiten beschrieben werden. Dabei werden die Daten eines Programms als Eigenschaften von Objekten und Funktionen bzw. Prozeduren als Methoden oder Fähigkeiten der Objekte in diesen zusammengefasst. Ein Programm kann somit als Sammlung von Objekten gelten, die mithilfe ihrer Methoden miteinander kommunizieren. Da auch Java objektorientiert ist, werden Sie dieses Konzept noch ausführlich kennenlernen.

1.2 Java

Bevor wir beginnen, intensiv mit Java und Eclipse zu arbeiten, möchte ich einen kurzen Einblick in die Entstehung und Entwicklung unserer Werkzeuge voranstellen. Das Wissen um die Motivation zur Entwicklung von Java und seine geschichtliche Entwicklung erleichtert das Verständnis an der einen oder anderen Stelle, wenngleich es für das Erlernen der Sprache nicht zwingend notwendig ist. Auf diese Einführungen möchte ich aber nicht verzichten. Sollten Sie den besonders schnellen Einstieg suchen, können Sie den ersten Teil dieses Abschnitts für spätere Mußestunden zurückstellen und direkt mit

Abschnitt 1.2.3 beginnen, denn dort werde ich die Installation und vorbereitende Arbeiten beschreiben.

1.2.1 Die Entstehungsgeschichte von Java

Was hat die Programmiersprache Java mit einer indonesischen Insel zu tun? Eigentlich gar nichts! Wie jede Neuentwicklung musste auch Java irgendwann einen Namen bekommen, und dabei war Java nicht unbedingt die erste Wahl. 1991 wurde bei *Sun Microsystems* eine Projektgruppe gegründet, die sich mit der künftigen Ausrichtung der Computer- und Softwareindustrie befassen sollte. Sie setzte sich zum Ziel, den Prototyp eines programmierbaren Geräts für die Steuerung von Haushaltsgeräten zu entwickeln.

Die erforderliche Software sollte klein und effizient, aber auch stabil und sicher sein. Vor diesem Hintergrund wurde eine neue Programmiersprache entwickelt, die objektorientiert war und sich zunächst stark an C++ orientierte. Der Leiter des Forschungsprojekts war *James Gosling*. Der Name der neuen Programmiersprache sollte *Oak* (*Object Application Kernel*) lauten. Inspiriert hatte Gosling der Anblick einer Eiche, die von einem Bürofenster aus zu sehen war. Als Ergebnis der inzwischen als *Green Project* bezeichneten Projektgruppe entstand ein kleines Gerät mit dem Namen **7* (*Star Seven*). Die Vermarktung des Geräts war nicht von Erfolg gekrönt. Geblieben sind von *7 nur *Duke*, ein kleines Männchen, das dem Benutzer im Display die Bedienung erleichtern sollte und das heute noch das Maskottchen von Java ist (siehe Abbildung 1.11), sowie die Programmiersprache, die für die Programmierung von *7 genutzt wurde.

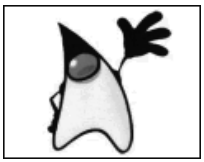


Abbildung 1.11 Duke, das Maskottchen von Java

Man erkannte, dass die kleine, objektorientierte, dabei plattformunabhängige und robuste Programmiersprache Oak sehr gut zum inzwischen massiv aufgekommenen World Wide Web passte. Somit stand plötzlich die Programmiersprache, die ursprünglich nur Teil eines Produkts sein sollte, selbst im Mittelpunkt der Entwicklung.

Da sich inzwischen herausgestellt hatte, dass der Name Oak geschützt war, musste ein anderer Name gefunden werden. Wahrscheinlich bei einer guten Tasse Kaffee einigte man sich auf den Namen *Java*. Die gleichnamige Insel verdankt diesen Gebrauch ihres Namens der Tatsache, dass auf ihrem Hochland ein exzellenter Kaffee angebaut wird. Folgerichtig ist das Logo von Java natürlich auch eine Kaffeetasse (siehe Abbildung 1.12).

Abbildung 1.12 Java-Logo

Die geringe Größe von Java-Anwendungen prädestinierte diese für den Download aus dem Internet. Das Potenzial war so offensichtlich, dass Java nur kurze Zeit später auch in die damals führenden Browser *Netscape Navigator* und *Internet Explorer* integriert wurde. Der 23.5.1995 wird von der Firma Sun Microsystems heute als die offizielle Geburtsstunde von Java angesehen. An diesem Tag wurde Java öffentlich vorgestellt, und *Marc Andreessen*, der Gründer von *Netscape*, kündigte die Integration von Java in Version 2.0 des Netscape Navigators an.

Die erste Version des *Java Development Kit* (*JDK1.0*) wurde 1996 veröffentlicht. Seit Frühjahr 2024 stehen Java 22 und eine Vielzahl spezieller *APIs* (*Application Programming Interfaces*) zur Verfügung. Ein API ist eine Programmierschnittstelle. Dem Programmierer werden dadurch Funktionen zur Verfügung gestellt, mit deren Hilfe er in eigenen Programmen auf ein bestimmtes System zugreifen kann. Bei diesen Systemen kann es sich um spezielle Hardware, aber auch um Softwarekomponenten wie z. B. eine grafische Oberfläche, ein Datenbanksystem oder ein soziales Netzwerk handeln.

Bei den Versionen von Java ist grundsätzlich Folgendes zu beachten: Neue Versionen von Java erscheinen derzeit in einem Abstand von einem halben Jahr. Sie unterscheiden sich allerdings stark bezüglich der Zeiträume, für die Support und Updates zur Verfügung gestellt werden. Alle zwei Jahre erscheint eine Version mit dem Zusatz *LTS* (*long-term support*). Die aktuellste LTS-Version ist das Java 21, das im Herbst 2023 erschienen ist. Alle anderen Versionen, die diesen Zusatz nicht tragen, erhalten lediglich für ein halbes Jahr Support und Updates.

1.2.2 Merkmale von Java

Wie ist Java in die Reihe der Programmiersprachen einzuordnen? Java ist eine Hochsprache. Verwendet sie nun aber einen Compiler oder einen Interpreter? Mit Java wurde ein neuer Weg beschritten, indem sowohl ein Compiler als auch ein Interpreter verwendet werden (siehe Abbildung 1.13). Nach der Erstellung des Quellcodes wird dieser von einem Compiler übersetzt. Allerdings wird der Quellcode nicht direkt in einen ausführbaren Code übersetzt, der auf dem System lauffähig ist, sondern in einen Zwischencode. Diese Zwischenstufe wird als *Bytecode* bezeichnet. Der Java-Compiler erzeugt beim Überset-

zen Bytecodedateien mit der Dateiendung *.class*. Dieser Umstand macht für die Ausführung des Programms einen Interpreter erforderlich, der allerdings keinen Quell-, sondern Bytecode interpretiert. Der Java-Interpreter, der auf dem betreffenden System installiert sein muss, wird entweder als *virtuelle Maschine* (*VM*) oder als *Java Runtime Environment* (*JRE*) bezeichnet.

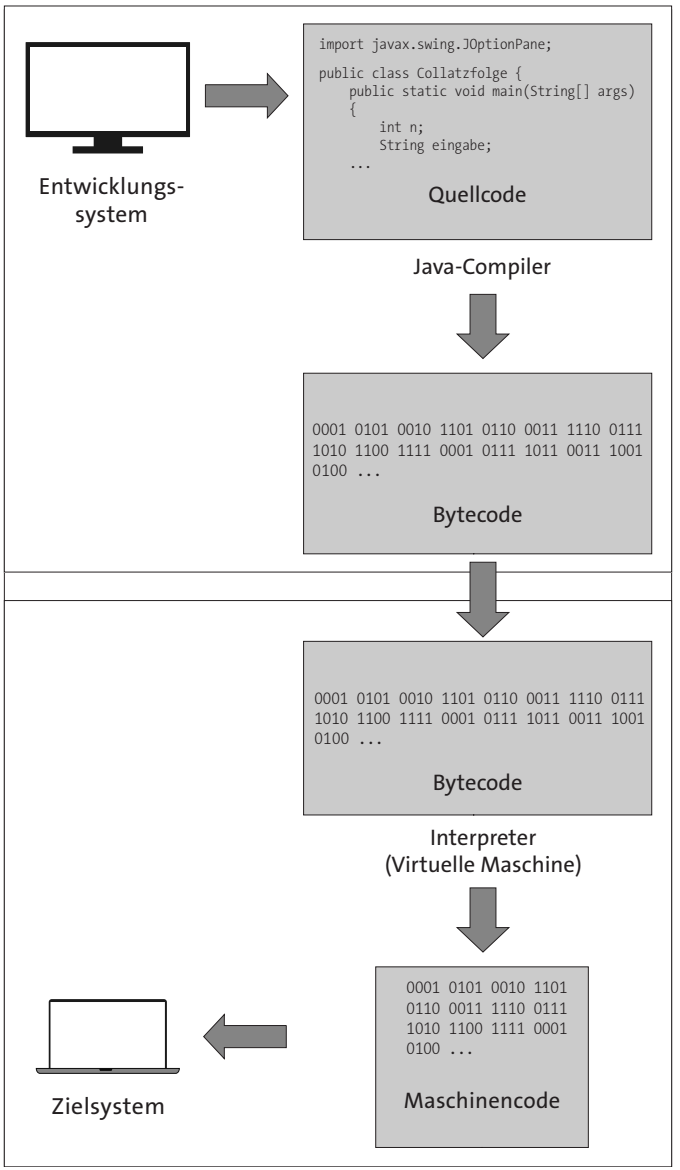


Abbildung 1.13 Hochsprachenprogrammierung mit Java

Java kombiniert die Vorteile eines Compiler- und eines Interpreter-Systems. Häufig wird die Aussage getätigt, dass sich durch die Verwendung eines Interpreters auf dem Zielsystem ein Geschwindigkeitsnachteil bei der Ausführung des Programms ergibt. Speziell für Java gilt jedoch, dass die Übersetzung des Bytecodes weit weniger zeitaufwendig als eine Übersetzung aus dem Quellcode ist. Zusätzlich wurde eine Vielzahl von Optimierungen vorgenommen, sodass der Geschwindigkeitsnachteil vor allem unter Berücksichtigung der heutigen Prozessorgeschwindigkeiten nicht mehr gravierend ausfällt. Der Einsatz von JIT-Compilern (Just-In-Time-Compilern) ermöglicht es sogar, Bytecode bei der Ausführung schneller als starren Maschinencode zu machen.

Dadurch, dass für alle gebräuchlichen Betriebssysteme Java-Compiler und virtuelle Maschinen verfügbar sind, besteht ein wesentlicher Vorteil der Programmiersprache Java in der *Plattformunabhängigkeit*. Dem Anwender ist häufig gar nicht bewusst, dass sein System über eine virtuelle Maschine verfügt. Diese wird oft mit der Installation eines Java-Programms installiert und steht dann allen Java-Anwendungen zur Verfügung. In vielen Fällen gelangt sie auch schon bei der Installation des Betriebssystems auf den Computer. Ob eine virtuelle Maschine auf Ihrem Rechner bereits verfügbar ist, werden Sie feststellen, wenn Sie versuchen, ein Java-Programm zu starten. Fehlt die virtuelle Maschine, werden Sie eine entsprechende Fehlermeldung erhalten. Unabhängig davon, ob Sie unter Windows, Linux oder macOS arbeiten, können Sie mit der Anweisung `java -version` abfragen, welche Java-Version installiert ist. Ist eine virtuelle Maschine installiert, erhalten Sie einen entsprechenden Hinweis auf das *Runtime Environment* (die *Laufzeitumgebung*). Dabei handelt es sich um den englischen Fachbegriff für die virtuelle Maschine. Unter Windows starten Sie für die Eingabe dieser Anweisung aus dem Startmenü die Anwendung Eingabeaufforderung (bzw. klicken dazu mit der rechten Maustaste auf den Start-Button), unter Linux und macOS verwenden Sie die Anwendung Terminal.

Wenn wir den Begriff *Java* verwenden, müssen wir grundsätzlich Folgendes beachten: Java besteht aus zwei Teilen: aus der Programmiersprache selbst, die in Bytecode übersetzt wird, und aus der JVM, auf der der Bytecode ausgeführt wird. Von der genauen Funktion der JVM brauchen Sie im Rahmen dieses Buches keine tiefe Kenntnis, Sie lernen aber die Programmiersprache Java. Es gibt inzwischen einen ganzen Strauß anderer Sprachen, die, um plattformunabhängig zu sein, ebenfalls in den JVM-Bytecode übersetzt werden, beispielsweise das weiter oben erwähnte *Kotlin* oder *Groovy*.

Java ist eine sehr universelle Programmiersprache, die sich für sehr viele unterschiedliche Einsatzbereiche eignet. Ihre Stärken kann sie vor allem dann ausspielen, wenn es darum geht, plattformunabhängige, robuste und sichere Programme zu erstellen. Sie eignet sich gut für verteilte Systeme und Netze und nicht zuletzt für internetbasierte Anwendungen. Nicht geeignet ist Java für sehr hardwarenahe Programme wie Treiber,

denn ein Java-Programm hat keine Möglichkeit, direkt auf die Hardware zuzugreifen. Alle Zugriffe auf die Hardware werden über die zwischengeschaltete virtuelle Maschine abgewickelt. Das Programm selbst hat keinen Einfluss darauf, wie auf die Hardware zugegriffen wird. Dies macht einerseits die Programme stabiler und plattformunabhängiger, führt aber gleichzeitig dazu, dass der Bereich der Treiberprogrammierung Sprachen wie C/C++ überlassen werden muss.

Java wurde sehr stark an C++ angelehnt, was sich noch in der ähnlichen Syntax zeigt. Aber Java hat auch einige Änderungen erfahren, weil die Entwickler Schwächen des Konzepts von C++ vermeiden wollten. Auf problematische Elemente wie Zeiger und die Mehrfachvererbung wurde verzichtet. Stattdessen wurde ein besonders sicheres Konzept der Speicherverwaltung implementiert.

Im Gegensatz zu anderen Programmiersprachen, die – wie oben erläutert – aus anderen, nicht objektorientierten Sprachen hervorgegangen sind, konnte Java ohne Rücksichtnahme auf »ererbte« Überbleibsel aus nicht objektorientierten Konzepten entwickelt werden. So wurde auf bestimmte Konzepte verzichtet, die wesentlich dazu beitragen, dass Programme fehleranfällig und unsicher werden.

Für den anhaltenden Erfolg von Java gibt es mehrere Gründe, von denen im Folgenden noch einmal zwei zusammenfassend genannt werden:

- Den Entwicklern von Java ist es gelungen, die Programmiersprache – wie sie es selbst formuliert haben – als »einfache, objektorientierte, verteilte, interpretierte, robuste, sichere, architekturneutrale, portable, performante, nebenläufige, dynamische« Sprache zu entwickeln.
- Die Entwicklungswerkzeuge und die Laufzeitumgebung für alle gängigen Prozessoren und Betriebssysteme sind frei verfügbar.

1.2.3 Installation von Java

Wenn von einer Java-Installation die Rede ist, muss grundsätzlich zwischen der Installation des JRE (Java Runtime Environment) oder des JDK (Java Development Kit) unterschieden werden. Die Installation des JRE, das auch, wie weiter oben beschrieben, als VM (virtuelle Maschine) bezeichnet wird, ist erforderlich, um Java-Programme auf dem betreffenden Computer ausführen zu können. Dabei handelt es sich um die Laufzeitumgebung und den Java-Interpreter. Damit allein können aber noch keine Java-Programme erstellt werden. Hierzu werden zusätzlich der Java-Compiler und eine Vielzahl von Bibliotheken benötigt, die in Packages organisiert sind. Wie oben erwähnt, wird für die Erstellung von Java-Programmen das JDK benutzt.

Diesem Buch liegt die Version 21 des *OpenJDK* zugrunde. Ich habe mich für diese Version entschieden, weil sie, wie weiter ober bereits beschrieben, die aktuellste LTS-Version (Long-Term-Support-Version) ist. Seit einiger Zeit steht das JDK nur noch als 64-Bit-Version zur Verfügung. Arbeiten Sie noch mit einem 32-Bit-System, müssen Sie entsprechend die 32-Bit-Version des JDK 8 installieren.

Seit der Übernahme von Sun Microsystems durch *Oracle* wird Java unter der Regie von Oracle weiterentwickelt. Oracle bietet seit Java 11 das JDK mit zwei unterschiedlichen Lizenzen an. Für das *OpenJDK* gilt die bekannte GNU General Public Licence v2 (GPLv2) mit einer sogenannten *Classpath-Exception* (CPE). Zusätzlich wird eine kommerzielle Lizenz für das Oracle JDK angeboten. Dieses JDK finden Sie unter der Adresse *https://oracle.com/java/technologies/downloads/*.

Mit der Veröffentlichung von Java 17 kann das Oracle JDK auch wieder für kommerzielle Zwecke kostenlos verwendet werden. Ob dieser neuerliche Schwenk in der Lizenzpolitik von Oracle von Dauer ist, bleibt abzuwarten. Für welche Lizenz man sich entscheiden sollte, hängt zurzeit nicht mehr vom Einsatzzweck ab (kommerziell oder nicht kommerziell), aber eventuell von der Notwendigkeit eines professionellen Supports und davon, ob Ihr Programm spezielle Oracle-Features verwendet, die über Java SE (Standard Edition) hinausgehen. Einen funktionalen Unterschied gibt es ab Version Java 11 laut einer Veröffentlichung des Oracle-Produktmanagements nicht. Das OpenJDK wird auch von zahlreichen anderen Anbietern zur Verfügung gestellt. Da ich in diesem Buch *Eclipse* als Entwicklungsumgebung verwenden werde, habe ich mich für *Eclipse Temurin* entschieden, das Sie über den Link *https://adoptium.net/de/temurin/releases/* für Ihr verwendetes Betriebssystem herunterladen können. Sie können ebensogut das OpenJDK eines anderen Anbieters verwenden. Von Oracle können Sie es über den Link *https://jdk.java.net* beziehen.

Wählen Sie den Link hinter der Angabe READY FOR USE: JDK22 bzw. die zum Zeitpunkt Ihres Downloads aktuelle Version aus (siehe Abbildung 1.14).

Da das JDK für unterschiedliche Betriebssysteme verfügbar ist, müssen Sie vor dem Download die Plattform auswählen, auf der die Installation erfolgen soll (siehe Abbildung 1.15).

Sie erhalten eine zum jeweiligen Betriebssystem passende Installationsdatei oder eine gepackte Datei (*.tar.gz-* bzw. *.zip*-Datei). Mit der Installationsdatei installieren Sie das JDK wie gewohnt unter Ihrem Betriebssystem. Haben Sie für den Download ein gepacktes Archivformat (*.tar.gz* oder *.zip*) gewählt, so finden Sie nach dem Entpacken der Datei einen Ordner mit dem Namen der entsprechenden Version (z. B. *jdk-21.0.3+9*).

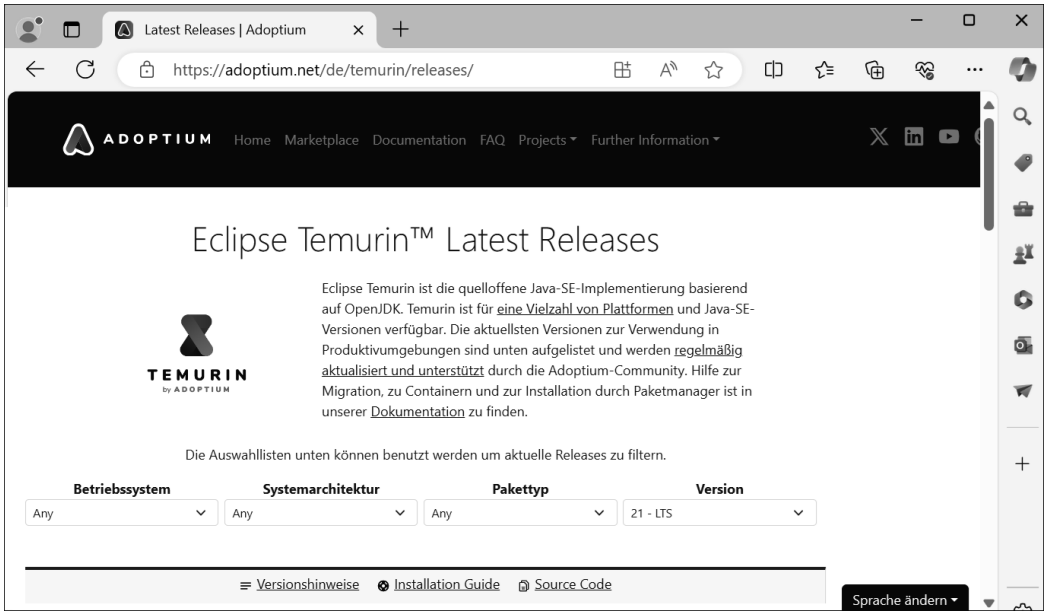


Abbildung 1.14 Download des OpenJDK Eclipse Temurin

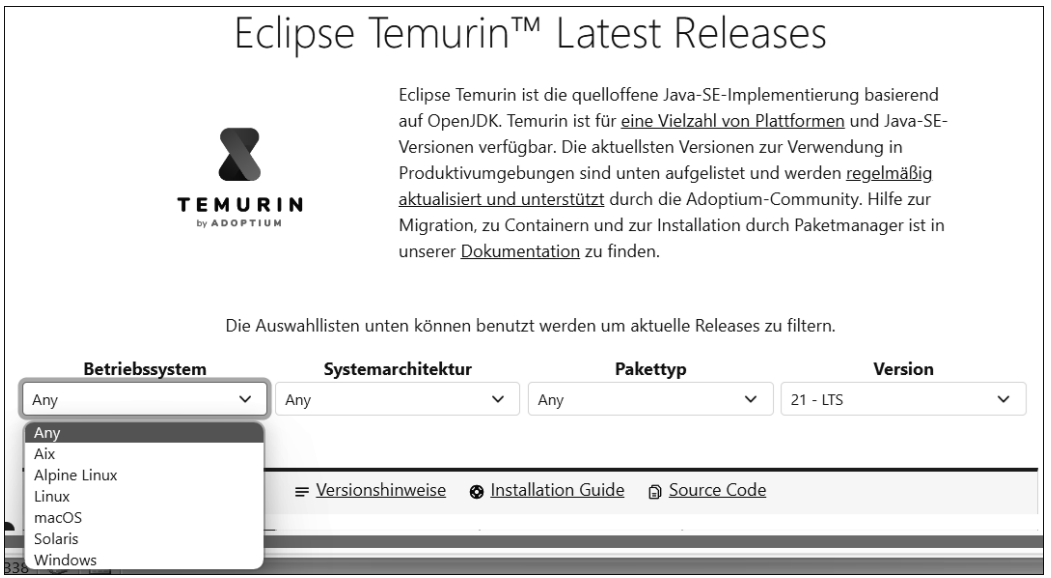


Abbildung 1.15 Auswahl der zum Betriebssystem passenden Version

Bei der Installation des JDK wird die Java-Laufzeitumgebung JRE mit installiert. Es handelt sich dabei um einen Minimalstandard, der erforderlich ist, um Java-Anwendungen laufen lassen zu können. Die JRE ist auf den meisten Rechnern bereits vorhanden, da sie von vielen anderen Programmen bei der Installation mit installiert wird. Sie sollten jedoch wissen, dass Java-Programme ohne JRE nicht gestartet werden können.

Zur Installation wird der entpackte Ordner in den jeweiligen Programmordner des Betriebssystems kopiert.

Das OpenJDK Eclipse Temurin wird unter Windows standardmäßig unterhalb von *C:\Programme\Eclipse Adoptium* installiert. Abbildung 1.16 zeigt die dabei erstellte Ordnerstruktur. Unter macOS wird der Ordner (*temurin-21.jdk*) in den Ordner */Library/Java/JavaVirtualMachines/* kopiert. Unter Linux, z. B. Ubuntu, kann es auch über den Package Manager mit der Anweisung `sudo apt-get install temurin` installiert werden. Oder aber Sie kopieren den entpackten Ordner (z. B. *jdk-21.0.3.9*) in das Verzeichnis */usr/lib/jvm/*.

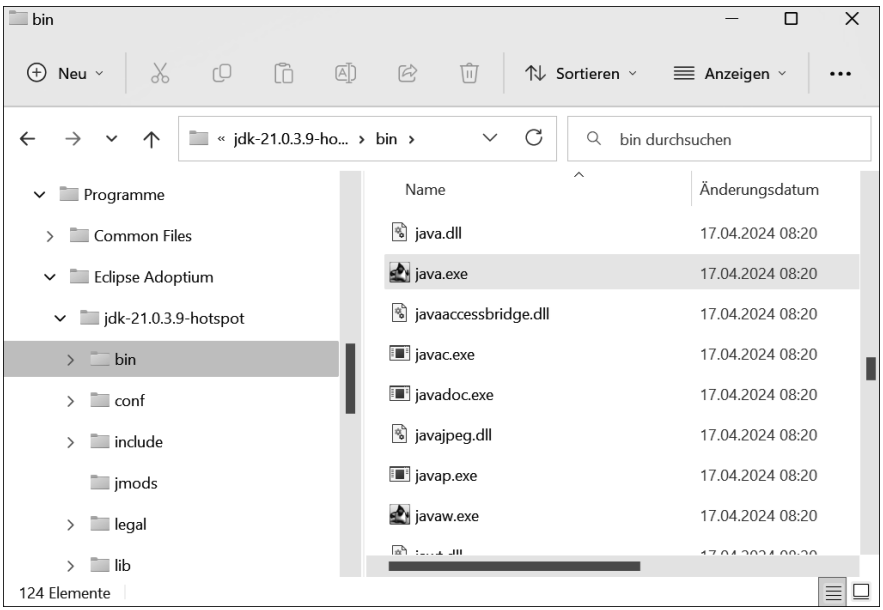


Abbildung 1.16 Ordnerstruktur des JDK

Es können durchaus auch mehrere JDK-Versionen nebeneinander existieren. Im rechten Bildausschnitt ist auszugsweise der Inhalt des Ordners *bin* zu sehen. Dieser Ordner enthält alle Binärdateien. Das sind ausführbare Dateien (Programme), unter denen sich u. a. auch der Java-Compiler *javac.exe* befindet.

Damit der Java-Compiler später möglichst einfach verwendet werden kann, müssen Sie unter Umständen Ihrem Betriebssystem den Installationsort bekannt machen. Sollten Sie zur Installation das Installationsprogramm für Ihr Betriebssystem (z. B. *.msi*-Datei für Windows, .pkg-Datei für macOS) bzw. den entsprechenden Paketmanager unter Linux verwendet haben, so hat dies die folgenden Schritte bereits für Sie erledigt. Dann können Sie mit Abschnitt 1.3, »Ein erstes Java-Programm«, fortfahren. Sie können die im Folgenden für Windows beschriebenen Schritte aber auch nachvollziehen, um zu überprüfen, welche Eintragungen die Installation vorgenommen hat.

Zum Starten eines Programms muss der Kommando-Interpreter des Betriebssystems wissen, wo sich ein Programm befindet. Wenn Sie ein Programm in dem Verzeichnis aufrufen, in dem Sie sich gerade befinden, wird der Kommando-Interpreter dieses finden. Ebenfalls unproblematisch ist es, eine vollständige Pfadangabe voranzustellen (z. B. *C:\programme\test*). Wenn wie hier beides nicht der Fall ist, wird das Programm nicht gefunden. Um das Problem zu beheben, verfügt jedes Betriebssystem über einen sogenannten Suchpfad, der in einer Umgebungsvariablen (das ist eine Variable des Betriebssystems) gespeichert ist. In einem Suchpfad können dem Betriebssystem alle Verzeichnisse bekannt gemacht werden, die der Kommando-Interpreter eines Betriebssystems in die Suche nach der Programmdatei einbeziehen soll. Zum Anpassen der Umgebungsvariablen überprüfen Sie zunächst, in welchem Verzeichnis das JDK bzw. die JRE installiert wurde (z. B. *C:\Programme\Eclipse Adoptium\jdk-21.0.3.9-hotspot*). Für die Pfadangabe ist noch der Unterordner *\bin* anzuhängen, zudem ist die englische Schreibweise für den Programmordner zu verwenden (also: *C:\Program Files\ Eclipse Adoptium\jdk-21.0.3.9-hotspot\bin*). Um die Umgebungsvariable dauerhaft zu setzen, gehen Sie unter Windows 10 folgendermaßen vor:

1. Öffnen Sie die Systemsteuerung über die Programmgruppe Windows-system.
2. Wählen Sie System und Sicherheit aus.
3. Wählen Sie System aus.
4. Wählen Sie Erweiterte Systemeinstellungen aus.
5. Betätigen Sie unter dem Reiter Erweitert die Schaltfläche Umgebungsvariablen (siehe Abbildung 1.17).

Unter Windows 11 wählen Sie in der Taskleiste das Lupensymbol für die Suche aus und geben als Suchbegriff *env* für Environment ein. Anschließend wählen Sie aus den Suchergebnissen »Systemumgebungsvariablen« aus.

Abbildung 1.17 Erweiterte Systemeigenschaften

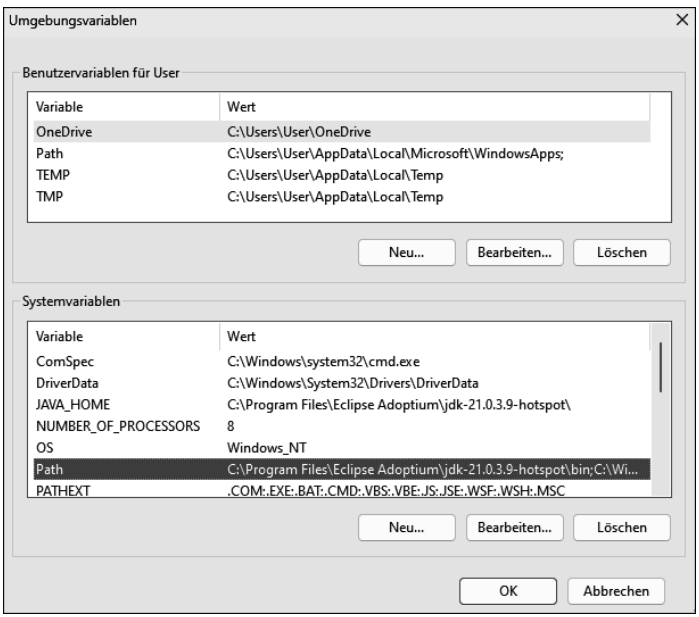


Abbildung 1.18 Umgebungsvariablen bearbeiten

Über die Schaltfläche Bearbeiten (siehe Abbildung 1.18) kann nun der Pfad erweitert werden. Sie ergänzen dazu die bestehenden Angaben um den Pfad zu Ihrer JDK-Installation so, wie oben beschrieben, ergänzt um den Unterordner *bin* (z. B. *C:\Program Files\Eclipse Adoptium\jdk-21.0.3.9-hotspot\bin* wie in Abbildung 1.19).

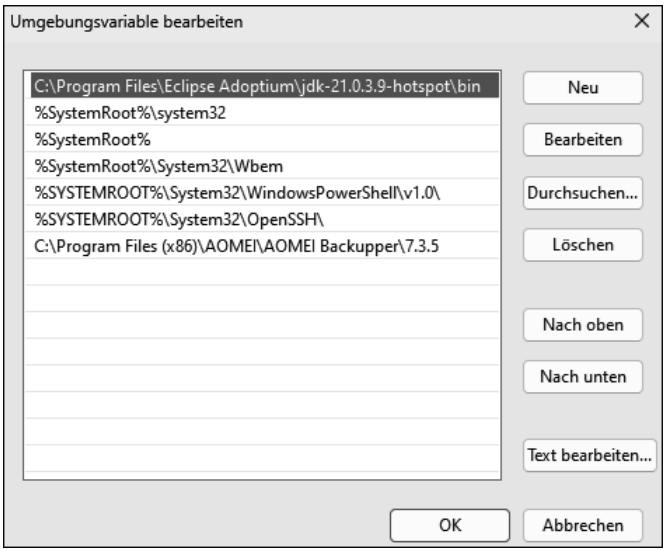


Abbildung 1.19 Path-Variable bearbeiten

1.3 Ein erstes Java-Programm

In diesem Abschnitt erstellen wir eine erste Anwendung. Dabei soll zunächst noch auf die Unterstützung durch eine Oberfläche wie Eclipse verzichtet werden, um zu zeigen, welche minimalen Voraussetzungen zur Erstellung einer Anwendung erforderlich sind. Außerdem werden bei dieser Vorgehensweise die Hintergründe der Java-Programmierung deutlich. Von der Programmerstellung bis zur Ausführung eines Java-Programms müssen immer drei Schritte durchlaufen werden:

1. Erstellen des *Quellcodes*
2. Kompilieren (Übersetzen) des *Quellcodes* in den *Bytecode*
3. Starten des Programms durch Übergeben des Bytecodes an den Java-Interpreter

Der Quellcode ist der Programmtext, den wir in der Programmiersprache Java schreiben. Den gespeicherten Programmcode erkennen Sie an der Dateinamenerweiterung *.java*. Der Bytecode ist ein Zwischencode, der für uns nicht mehr ohne Weiteres ver-

ständlich ist. Er ist in der Bytecodedatei mit der Dateinamenerweiterung *.class* gespeichert und kann vom Java-Interpreter ausgeführt werden. Wie bereits erwähnt, ist der Java-Interpreter ein wesentlicher Bestandteil der virtuellen Maschine. Um ein Java-Programm zu starten, muss diese virtuelle Maschine (JRE) auf dem entsprechenden Computer installiert sein. Der Java-Compiler wird nur vom Programmersteller (Programmierer) benötigt und ist Bestandteil des JDK.

1.3.1 Die Arbeitsumgebung vorbereiten

Bevor wir die oben genannten drei Schritte zum ersten Mal durchlaufen, bereiten wir uns eine Arbeitsumgebung vor.

Es ist grundsätzlich zu empfehlen, eine Verzeichnisstruktur als Arbeitsumgebung zu erstellen. Bei der Arbeit mit Eclipse wird diese automatisch als sogenannte *Workbench* erstellt. In dieser werden dann alle Dateien gespeichert, die bei der Programmerstellung erzeugt werden. Wie es sich allgemein in der Programmierung durchgesetzt hat, sprechen wir bei unserer Arbeit von *Projekten*. Ein Projekt umfasst alle Dateien, die zu einem Programm gehören. Zum Beispiel kann eine Dokumentation und alles, was der Programmierer sonst noch als notwendig oder sinnvoll erachtet, hinzukommen.

Merke

Ein *Projekt* umfasst alle Dateien, die zur Realisierung eines Programms notwendig bzw. hilfreich sind. In der Regel sollten Sie alle diese Dateien in einem gemeinsamen Projektordner verwalten.

Die meisten Entwicklungsumgebungen legen einen übergeordneten Ordner an, unter dem alle Programmierprojekte gespeichert werden. Eclipse verwendet für diesen Ordner den Namen *Workbench*. Auch wenn wir die Entwicklungsumgebung im Augenblick noch nicht verwenden, werden wir bereits jetzt diese Vorgehensweise anwenden.

Legen Sie also nun auf einem Laufwerk Ihrer Wahl einen Ordner mit dem Namen *Java* an. In diesem Buch wird durchgängig Laufwerk *H:* verwendet. Erstellen Sie Abbildung 1.20 entsprechend einen weiteren Unterordner mit dem Namen *Programme*. Unterhalb dieses Ordners werden wir unsere Programmierprojekte speichern. Als ersten Projektordner legen Sie *JavaUebung01* an. (Die Übungsnummern sollen die Zuordnung zu den Kapiteln erleichtern.)

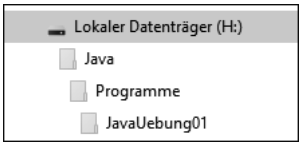


Abbildung 1.20 Ordnerstruktur der Arbeitsumgebung

Auch wenn es eigentlich unproblematisch wäre, den Ordnernamen mit dem Umlaut zu schreiben, sollten Sie sich an die englische Schreibweise halten und auf Umlaute für die Dateinamen und auch für die Bezeichner innerhalb des Java-Programms verzichten. Wenn Sie sich diese Vorgehensweise angewöhnen, werden Sie sich auch beim Umstieg auf andere Umgebungen und Programmiersprachen leichter tun, denn in vielen Programmiersprachen sind keine Umlaute in Bezeichnern zulässig: Sie führen zu Übersetzungsfehlern und müssen vollständig ausgeräumt werden, damit der Compiler oder Interpreter den Programmtext in ein lauffähiges Programm übersetzen kann.

1.3.2 Wie sind Java-Programme aufgebaut?

Ein Java-Programm besteht immer aus einer oder mehreren *Klassen* (`class`). Der Programmtext einer Klasse sollte jeweils in einer eigenen Datei gespeichert werden. Diese Programmtextdateien müssen die Dateiendung *.java* haben und im selben Ordner gespeichert werden (siehe Abbildung 1.21).

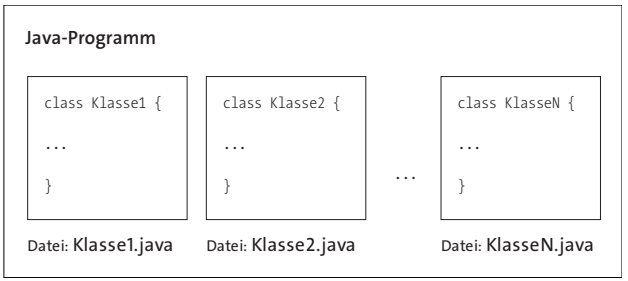


Abbildung 1.21 Aufbau eines Java-Programms

Merke

Der Name der Klasse und der Name der Datei, in der die Klasse deklariert ist, müssen identisch sein. Es gibt allerdings eine Ausnahme. Wenn Sie eine *.java*-Datei direkt mit `java` ausführen, was noch gezeigt wird, kann der Dateiname vom Klassennamen abweichen. Ansonsten müssen Sie die Groß- und Kleinschreibung beachten. Klassennamen sollten immer mit einem Großbuchstaben beginnen.

Wir wollen zunächst vom einfachsten Fall ausgehen. Das gesamte Java-Programm besteht dann aus einer einzigen Klasse (siehe Abbildung 1.22). Beachten Sie, dass eine Klasse nur dann als Programm ausgeführt werden kann, wenn sie über eine `main`-Methode verfügt. Der Begriff *Methode* spielt in der objektorientierten Programmierung eine ganz entscheidende Rolle. In Abschnitt 5.3 werden wir uns sehr ausführlich mit Methoden auseinandersetzen.

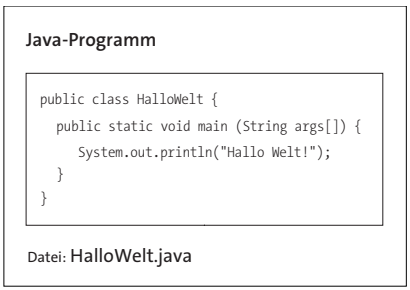


Abbildung 1.22 Ein einfaches Java-Programm

Die Klasse `HalloWelt` wird mit folgendem Rahmen definiert:

```
public class HalloWelt {
}
```

Listing 1.2 Deklaration einer Klasse

Zwischen den geschweiften Klammern ist nun die Klasse näher zu beschreiben. Soll die Klasse ein ausführbares Programm darstellen, muss dort eine `main`-Methode stehen. Diese wird mit der Zeile `public static void main(String[]args)` eingeleitet:

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

Listing 1.3 Der Rahmen der »main«-Methode

Merke

Die `main`-Methode bildet immer den Einstiegspunkt in ein Java-Programm. Eine Klasse eines Java-Programms muss deshalb auf jeden Fall eine `main`-Methode besitzen, damit sie als Programm gestartet werden kann.

Zwischen den geschweiften Klammern stehen nun die Programmbefehle (*Anweisungen*), die beim Programmstart ausgeführt werden sollen. Unter einer Anweisung versteht man eine in der jeweiligen Programmiersprache formulierte einzelne Vorschrift,

die im Rahmen der Abarbeitung des Programms auszuführen ist. Wie Sie hier sehen, werden die geschweiften Klammern verwendet, um zusammengehörige Programmteile zu einem sogenannten *Block* zusammenzufassen. Sie legen hier z. B. eindeutig den Anfang und das Ende der Klassendefinition bzw. der `main`-Methode fest.

Merke

Mit geschweiften Klammern `{}` können Sie zusammengehörige Programmteile zu einem *Block* zusammenfassen.

In dem Beispiel aus Abbildung 1.22 enthält die `main`-Methode nur eine einzige Anweisung. Mit der Anweisung `System.out.println` wird eine Textzeile ausgegeben. Welcher Text ausgegeben wird, wird hinter der Anweisung in runden Klammern angegeben. Der Text, der dort zwischen Anführungszeichen eingetragen ist, wird in exakt der gleichen Schreibweise ausgegeben. Abgeschlossen wird jede Anweisung mit einem Semikolon (;). Neben der Anweisung `System.out.println` steht Ihnen auch die Anweisung `System.out.print` zur Verfügung. Dadurch, dass bei ihr die beiden Buchstaben `ln` fehlen (`ln` ist die Abkürzung für das englische Wort »line«), wird die ausgegebene Textzeile nicht abgeschlossen. Damit wird eine folgende Ausgabe in der gleichen Zeile weitergeschrieben. Nach einer mit `println` abgeschlossenen Zeile wird hingegen eine sich anschließende Ausgabe in einer neuen Zeile erfolgen.

Merke

Jede Anweisung wird mit einem Semikolon (;) abgeschlossen.

```
public class HalloWelt {
   public static void main (String[] args) {
       System.out.println("Das ist eine erste Zeile!");
       System.out.print("Anfang der zweiten Zeile ");
       System.out.println("und Fortsetzung von Zeile 2!");
       System.out.println("Das ist die dritte Zeile!");
   }
}
```

Listing 1.4 Einsatz von »print« und »println«

Hinweis

Wenn Sie einzelne Anweisungen ausprobieren wollen, können Sie dazu die *interaktive Java-Shell* (*JShell*) verwenden. Dort können Sie direkt Java-Anweisungen eingeben und

deren Ergebnis beobachten, ohne eine **.java*-Datei erstellen, kompilieren und ausführen zu müssen. Dies erlaubt *exploratives Programmieren* – oder zu gut Deutsch: ausprobieren. Das Programm liegt im *bin*-Ordner Ihrer Java-Installation, und Sie starten die Shell einfach in der Eingabeaufforderung bzw. in einem Terminal durch die Eingabe von `jshell`. Hinter dem Befehlsprompt `jshell>`, der nun erscheint, können Sie dann die zu testenden Java-Anweisungen eingeben und ausprobieren. Sie beenden die Java-Shell durch die Eingabe von `/exit`. Eine kurze Einführung in die Verwendung der JShell finden Sie unter *https://javatutorial.net/java-9-jshell-example*.

1.3.3 Schritt für Schritt zum ersten Programm

Sie haben nun den grundsätzlichen Aufbau eines Java-Programms kennengelernt. Das Hallo-Welt-Programm, das Sie jetzt selbst schreiben sollen, können Sie als Minimalprogramm erstellen. Es besteht nur aus einer Klasse mit einer einzigen Methode mit dem Namen `main`.

Dieses Programm, das fast immer am Beginn eines Programmierkurses steht, ist ein Minimalprogramm, das sich dadurch bemerkbar machen soll, dass es den einfachen Gruß »Hallo Welt!« ausgibt.

1. Schritt: Den Quellcode erstellen

Da wir zunächst noch auf Eclipse verzichten, öffnen Sie einen einfachen Editor, wie er etwa unter Windows in der Programmgruppe Windows-Zubehör unter dem Namen Editor zu finden ist. Linux-User werden von den zahlreich vorhandenen Editoren vielleicht *joe*, *vi* oder *gedit* nutzen, während Mac-User auf den Editor *TextEdit* zurückgreifen werden. Hier geben Sie den folgenden Quellcode ein. Speichern Sie die Datei unter dem Namen *HalloWelt.java*. Achten Sie dabei aber genau auf Groß- und Kleinschreibung. Wenn Sie einen Editor verwenden, der unterschiedliche Formatierungen erlaubt, müssen Sie unbedingt darauf achten, dass Sie den Programmtext als reinen Text ohne Formatierungen speichern. So müssen Mac-User, die *TextEdit* verwenden, den Menüpunkt TextEdit • Einstellungen aufrufen, auf Reiner Text umstellen und das Häkchen bei Intelligente Anführungszeichen entfernen.

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

Listing 1.5 Der Quellcode des Hallo-Welt-Programms

Ich möchte Ihnen bereits hier einige formale Richtlinien ans Herz legen. Es ist für die Funktion des Programms vollkommen unerheblich, auf wie viele Zeilen Sie Ihren Programmcode verteilen. Für die Lesbarkeit und damit für die Nachvollziehbarkeit des Programmcodes ist die Formatierung aber sehr wichtig. Sie werden sich bei der Arbeit mit Java immer wieder Anregungen und Hilfestellungen von Programmcodes anderer Programmierer holen und auf die zahlreich vorhandenen Tutorials im Internet zurückgreifen. Es vereinfacht die Einarbeitung in fremden Programmcode wesentlich, wenn sich alle an die gleichen Formatierungsrichtlinien halten. Sie finden die Formatierungsrichtlinien der Java-Entwickler an zentraler Stelle im Internet unter folgender Adresse:

www.oracle.com/technetwork/java/javase/documentation/codeconvtoc-136057.html

Als *.pdf*-Datei finden Sie die *Code Conventions* unter diesem Link:

www.oracle.com/technetwork/java/codeconventions-150003.pdf

Es wäre durchaus denkbar, den Programmtext folgendermaßen einzugeben, und es lassen sich auch durchaus Argumente für eine solche Formatierung finden. So würden z. B. öffnende und schließende Klammern, die zusammengehören, weil sie einen Block umschließen, untereinander stehen:

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

Listing 1.6 Alternative Formatierung des Quellcodes

Die Code Conventions der Java-Entwickler geben vor, dass öffnende Klammern am Ende der Zeile stehen sollten, die den Block einleitet. In den folgenden Zeilen werden Einrückungen vorgenommen, bis die schließende Klammer wieder auf der gleichen Höhe wie die einleitende Zeile positioniert wird. Dadurch stehen die schließenden Klammern immer unter den Ausdrücken, die den Block einleiten. Der Programmcode wird dadurch etwas kompakter und trotzdem gut strukturiert.

Als Maß für die Einrückungen geben die Code Conventions vier Leerstellen vor. Aus Platzgründen wird in diesem Buch an der einen oder anderen Stelle ein wenig von dieser Vorgabe abgewichen. Das hat aber ausschließlich satztechnische Gründe. Abbildung 1.23 zeigt die Struktur des Quellcodes im Editor.

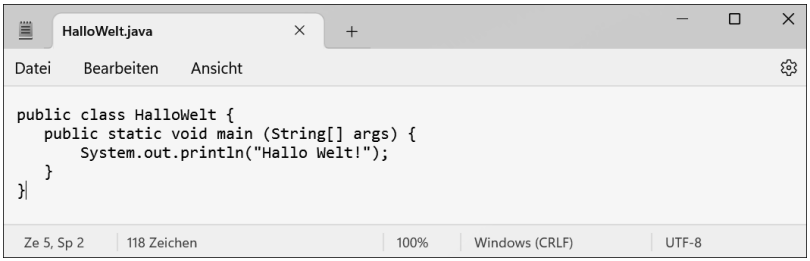


Abbildung 1.23 Hallo-Welt-Quellcode im Editor

Den Quellcode speichern Sie unter dem Namen *HalloWelt.java* in dem Ordner *H:\Java\Programme\JavaUebung01*, den wir in Abschnitt 1.3.1, »Die Arbeitsumgebung vorbereiten«, angelegt haben.

Hinweis

Der Windows-Editor hängt beim Speichern grundsätzlich die Erweiterung *.txt* an den Dateinamen an. Dies ist im Dialog Speichern bzw. Speichern unter am eingestellten Dateityp zu erkennen. Unser Java-Quellcode muss aber die Erweiterung *.java* verwenden. Wenn Sie als Dateityp Textdateien (*.txt) eingestellt lassen, wird unsere Datei unter dem Namen *HalloWelt.java.txt* gespeichert. Damit kein *.txt* angehängt wird, ändern Sie die Einstellung des Dateityps in Alle Dateien ab (siehe Abbildung 1.24).

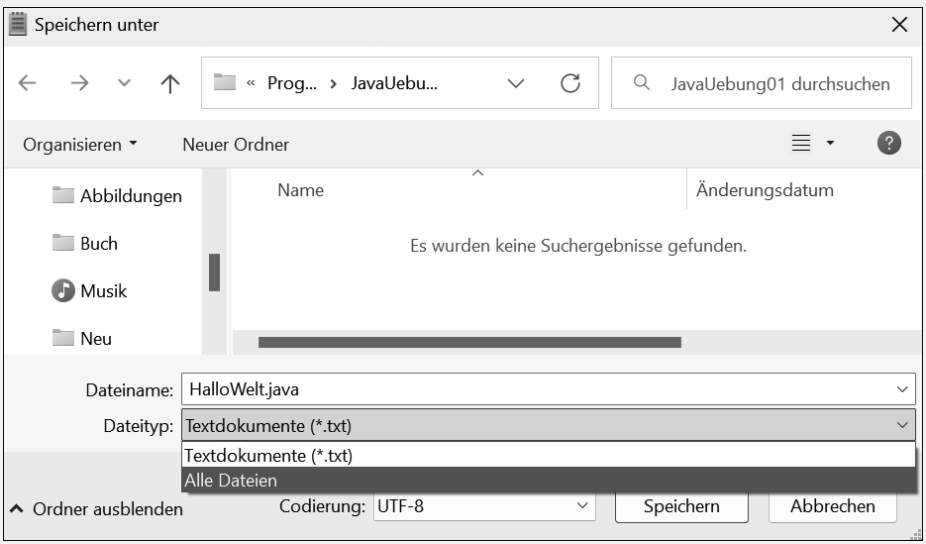


Abbildung 1.24 Den Dateityp im Editor festlegen

Dadurch wird beim Speichern exakt der von Ihnen eingetragene Dateiname verwendet. Achten Sie auch darauf, dass unter Codierung UTF-8 eingestellt ist. Häufig ist dort ANSI voreingestellt. Sollte dies der Fall sein, ändern Sie auf UTF-8, sonst laufen Sie Gefahr, dass bei der Verwendung internationaler Zeichen, wie z. B. Umlauten, Probleme auftreten.

Der Quellcode unseres Programms ist damit fertiggestellt.

2. Schritt: Den Quellcode in Bytecode übersetzen

Im folgenden Schritt übergeben Sie den Quellcode an den Java-Compiler. Dieser übersetzt den Quellcode in den sogenannten Bytecode, der für den Java-Interpreter verständlich ist. Für diesen Schritt starten Sie die Eingabeaufforderung über die Suchfunktion von Windows. Unter macOS entspricht dieses Programm dem TERMINAL, das Sie über FINDER • PROGRAMME • DIENSTPROGRAMME starten können. Linux-Usern steht das TERMINAL je nach Distribution meist unter ANWENDUNGEN • ZUBEHÖR zur Verfügung. Alternativ können Sie die Eingabeaufforderung auch über START • PROGRAMME/DATEIEN DURCHSUCHEN starten (siehe Abbildung 1.25).

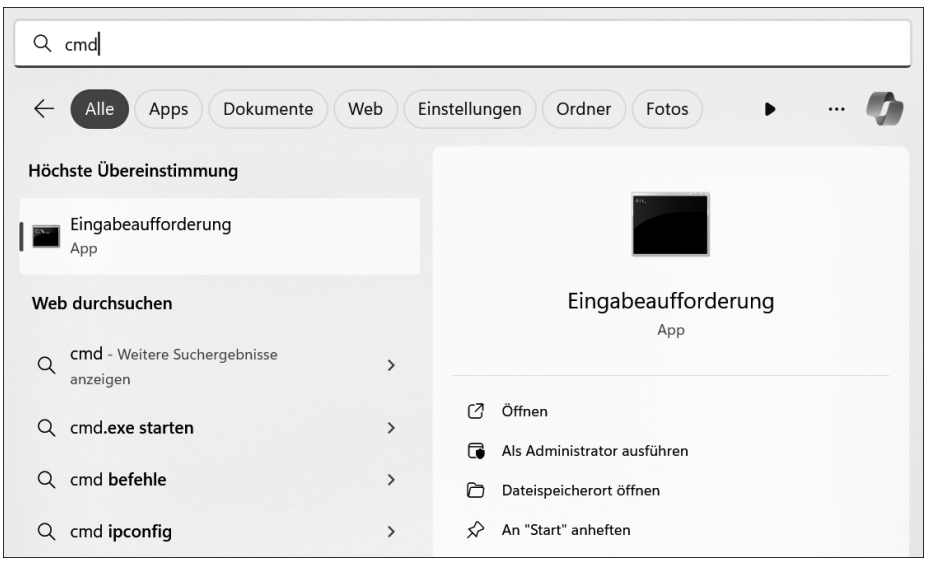


Abbildung 1.25 Sie finden die Eingabeaufforderung über die Suchfunktion von Windows 10 bzw. Windows 11.

Geben Sie hier `cmd` als Suchbegriff ein. Daraufhin erscheint im Startfenster unter PROGRAMME das Startsymbol der Eingabeaufforderung (siehe Abbildung 1.26). Die Eingabeaufforderung wird durch Aufruf der Datei *cmd.exe* gestartet. Diese befindet sich bei

einer Standardinstallation von Windows im Ordner *C:\windows\system32*. Die Suchfunktion erleichtert Ihnen hier das Auffinden erheblich.

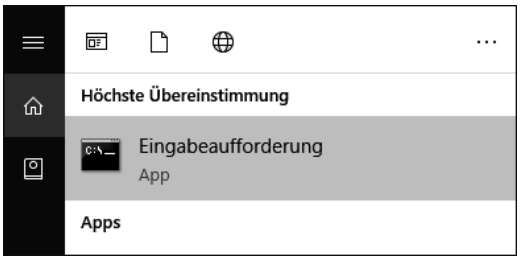


Abbildung 1.26 Das Programmsymbol der Eingabeaufforderung

Durch den Abschluss der Eingabe mit der ↵-Taste oder mit einem Mausklick auf den Eintrag unter HÖCHSTE ÜBEREINSTIMMUNG starten Sie die Eingabeaufforderung (siehe Abbildung 1.27).

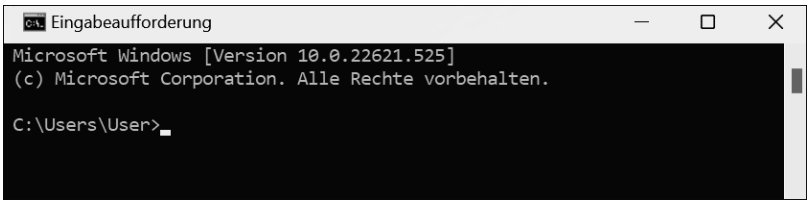


Abbildung 1.27 Die Eingabeaufforderung unmittelbar nach dem Start

Die Eingabeaufforderung ist eine *Textkonsole*, die als sogenannten *Befehlsprompt* eine Pfadangabe verwendet (siehe Abbildung 1.28). Diese Pfadangabe stellt den momentanen Aufenthaltsort dar. Standardmäßig wird hier zunächst der Stammordner des angemeldeten Benutzers verwendet. Wechseln Sie in den Ordner *JavaUebung01*, in dem auch unser Quellcode des Hallo-Welt-Programms gespeichert wurde. Mit folgenden Eingaben, die jeweils mit ↵ bestätigt werden, wechseln Sie in den Projektordner:

```
H:
cd Java\Programme\JavaUebung01
```

Mit der Eingabe `H:` machen Sie das Laufwerk *H:* zum aktuellen Laufwerk. Mit der Anweisung `cd` (*change directory*) wechseln Sie in unseren Projektordner, indem Sie hinter `cd` den Pfad zu diesem Ordner angeben. Der Befehlsprompt zeigt Ihnen anschließend auch den entsprechenden Pfad an (siehe Abbildung 1.28). Hinter dem Befehlsprompt können Sie nun weitere Anweisungen eingeben. Zur Kontrolle geben Sie das *DOS-Kommando* `dir` ein. Es listet uns den Inhalt des Ordners auf (siehe Abbildung 1.29). Darin dürfte sich zu diesem Zeitpunkt nur der gespeicherte Quellcode als Datei *HalloWelt.java* befinden.

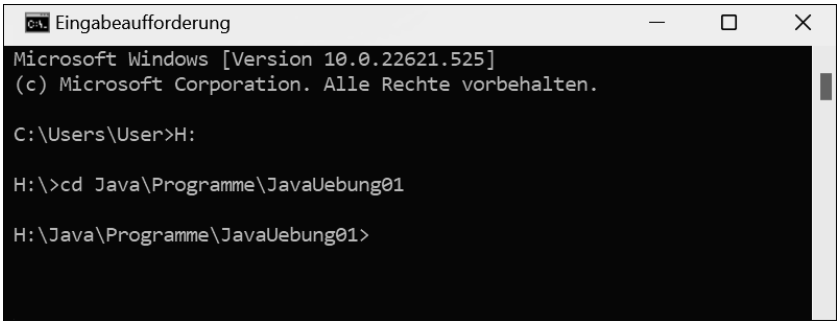


Abbildung 1.28 Die Eingabeaufforderung nach dem Ordnerwechsel

Tatsächlich werden neben der Datei *HalloWelt.java* noch zwei Ordner mit der Kennung *<DIR>* angezeigt (*DIR* steht für *Directory* und entspricht unserem *Ordner*). Hinter diesen beiden Ordnern mit den eigentümlichen Namen ».« und »..« verbergen sich aber lediglich zwei Verweise auf Ordner, die das Betriebssystem zur Navigation im Ordnersystem benötigt.

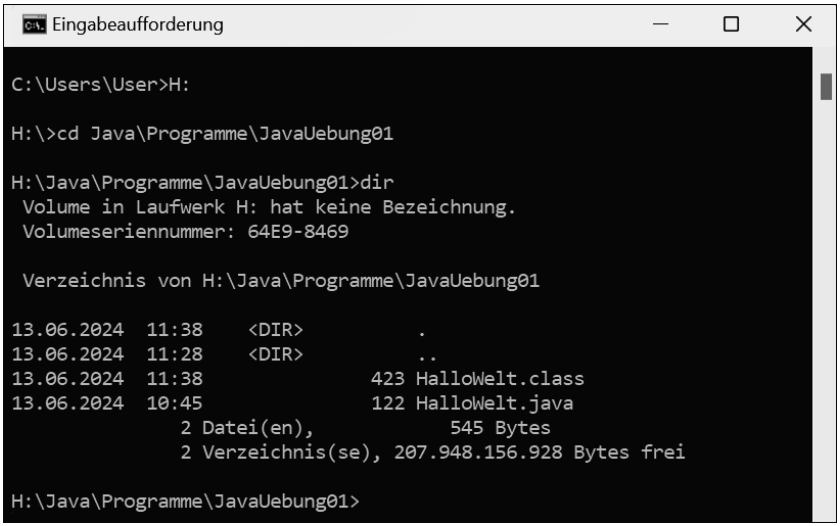


Abbildung 1.29 Die Eingabeaufforderung mit Ausgabe des Ordnerinhalts

Nun übergeben Sie unseren Programmtext an den Java-Compiler, damit dieser durch Übersetzen den Bytecode erstellt. Mit der Anweisung `javac`, gefolgt vom Namen der zu übersetzenden Datei, starten Sie den Übersetzungsvorgang (siehe Abbildung 1.30):

```
javac HalloWelt.java
```

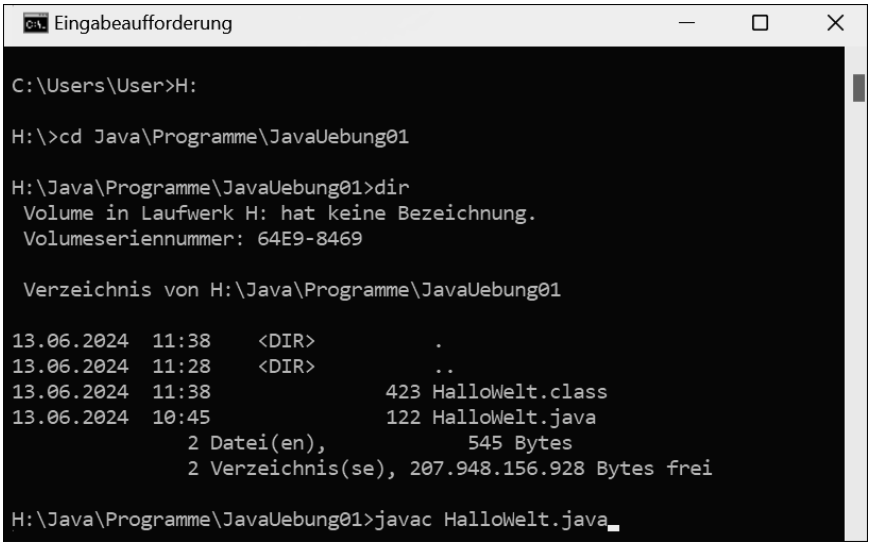


Abbildung 1.30 Java-Compiler aufrufen

Bis der Befehlsprompt wieder erscheint, wird ein kurzer Augenblick vergehen – dieser entspricht der Zeit, die der Compiler zum Übersetzen und zur Speicherung des Bytecodes benötigt. Und hier gilt die Aussage: »Keine Nachrichten sind gute Nachrichten.« Falls hier Meldungen erscheinen, handelt es sich um Fehlermeldungen oder Warnungen. Das bedeutet, dass unser gespeicherter Quellcode noch Fehler enthält und korrigiert werden muss, bevor der Bytecode erstellt werden kann.

Mit einem erneuten `dir` können Sie kontrollieren, ob der Bytecode erstellt wurde. Sollte dies der Fall sein, finden Sie jetzt zusätzlich die Datei *HalloWelt.class* in unserem Ordner (siehe Abbildung 1.31).

Abbildung 1.32 zeigt exemplarisch die Konsolenausgabe, wenn der Übersetzungsvorgang wegen eines Fehlers im Quellcode nicht durchgeführt werden konnte.

Der Ausdruck `' ; ' expected` gibt einen Hinweis auf die Art des Fehlers. Hier trifft der Hinweis genau zu, denn im Quellcode wurde ein Semikolon vergessen. Mit dem Caret-Symbol (^) wird eine Position im Quellcode markiert, an der der Fehler wahrscheinlich vorliegt. In diesem Beispiel liegt der Compiler genau richtig. Das sieht nicht immer so optimal aus; häufig ist der Fehler auch in einer anderen Zeile zu suchen. Schließlich wird mit der zusammenfassenden Angabe `1 error` angezeigt, auf wie viele Fehler der Compiler beim Übersetzungsversuch gestoßen ist. Lassen Sie sich von dieser Zahlenangabe nicht entmutigen, auch wenn sie sehr groß ausfallen sollte.

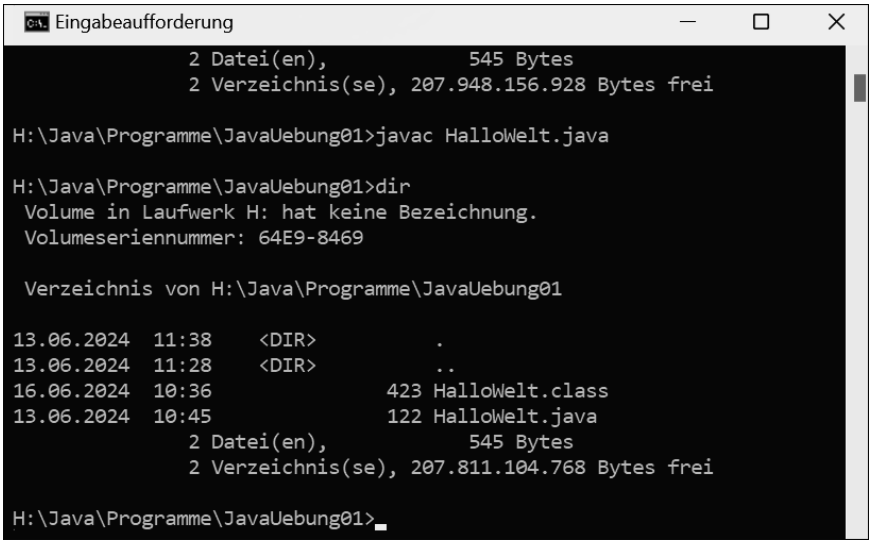


Abbildung 1.31 Inhaltsverzeichnis nach der Übersetzung

Abbildung 1.32 Übersetzungsversuch mit Fehlermeldung

3. Schritt: Den Bytecode an den Java-Interpreter zur Ausführung übergeben

Die Übergabe des Bytecodes zur Ausführung des Programms erfolgt mit der folgenden Tastatureingabe (siehe Abbildung 1.33):

```
java HalloWelt
```

Der Eingabe ist zu entnehmen, dass der Java-Interpreter *java* (genau genommen *java.exe*) heißt und dass als Parameter der Klassenname des Bytecodes übergeben wird. Die Interpreter-Datei *java.exe* ist übrigens, ebenso wie der Compiler, im Ordner *bin* des JDK zu finden. Achten Sie darauf, dass beim Namen der Bytecodedatei keine Dateinamenerweiterung (*.class*) angegeben werden darf.

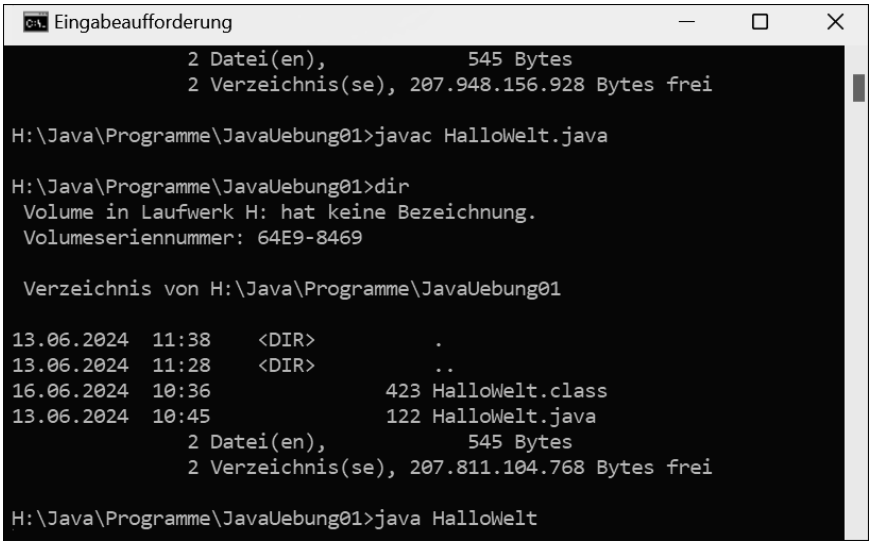


Abbildung 1.33 Programmstart durch Aufruf des Interpreters

Als Ergebnis sehen Sie die Ausgabe des Programms, also folgender Anweisung (siehe Abbildung 1.34):

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

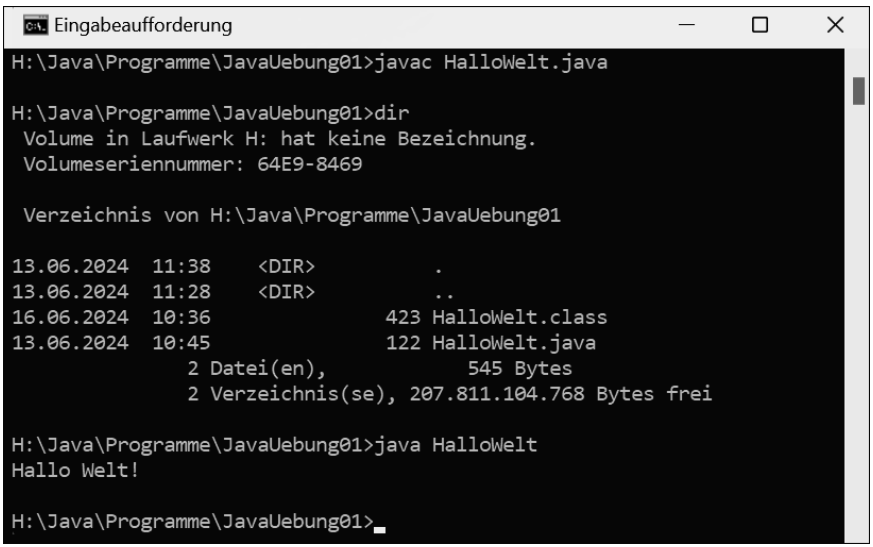


Abbildung 1.34 Ausgabe des Hallo-Welt-Programms

Nun haben Sie die drei Schritte für unser erstes Programm durchlaufen. Sie konnten an ihm die wesentlichen Bestandteile erkennen, die Sie für die Programmierung mit Java benötigen. Darüber hinaus konnten Sie feststellen, dass Sie immer wieder die Anwendung wechseln mussten: Zunächst haben Sie Quellcode in einem Editor erstellt, dann mussten Sie in die Eingabeaufforderung wechseln, um den Quelltext zu übersetzen und den Bytecode auszuführen.

In der Regel werden Sie nach der Übersetzung sehr häufig wieder zum Editor zurückkehren müssen, weil ein Programm nach der ersten Eingabe selten bereits fehlerfrei sein wird. Nach jeder Korrektur müssen Sie durch eine erneute Übersetzung überprüfen, ob der Quelltext nun fehlerfrei ist, und dann den gesamten Ablauf eventuell mehrmals wiederholen.

1.3.4 Single-File-Source-Code-Programme

Ich habe Ihnen nun gezeigt, wie Sie die drei Schritte vom Erstellen des Quellcodes bis zur Ausführung eines Java-Programms durchführen. Diese drei Schritte werden immer vollzogen und stellen damit den allgemeingültigen Ablauf dar. Ich möchte Ihnen aber nicht vorenthalten, dass unter Umständen die beiden Schritte »Kompilieren in den Bytecode« und »Ausführen des Programms« zusammengefasst werden können.

Befindet sich der gesamte Quellcode Ihres Java-Programms in einer einzigen Datei (*Single-File-Source-Code-Programm*), können Sie den zweiten Schritt (den Quellcode in Bytecode übersetzen) überspringen. Sie rufen dazu den Java-Interpreter *java* auf und übergeben ihm nicht den Bytecode (der ja noch nicht erzeugt ist), sondern die Quellcodedatei. Für unser Hallo-Welt-Programm geben Sie in der Eingabeaufforderung `java HalloWelt.java` ein. Der zweite Schritt zur Erstellung des Bytecodes entfällt dadurch nicht, er wird aber automatisch ausgeführt, bevor der Interpreter das Programm ausführt. Es ist also eine Arbeitserleichterung für Sie, da Sie den Compiler nicht händisch aufrufen müssen.

Da es sich bei den meisten unserer Programme um Single-File-Source-Code-Programme handelt, können Sie diese Möglichkeit bei der weiteren Arbeit intensiv nutzen.

1.4 Übungsaufgaben

Nun folgen einige Übungsaufgaben, die Sie dazu nutzen können, die oben erläuterten Abläufe zu trainieren. Die Fehler, die bei der Eingabe mit ziemlicher Sicherheit auftreten werden, korrigieren Sie mithilfe der vom Compiler erzeugten Fehlermeldungen. Da

noch keine Grundlagen der Programmiersprache behandelt wurden, sind die Quellcodes der Übungsaufgaben vollständig vorgegeben und müssen nur abgeschrieben werden. Sie sollen schließlich noch keine Java-Programme entwickeln, sondern zunächst die Handhabung von Compiler und Interpreter üben. Gleichzeitig sollen einige typische Features (wie die Parameterübergabe beim Programmstart und die Ausgabe grafischer Fenster) demonstriert werden. Verständnisprobleme, die den Quellcode betreffen, werden hier bewusst in Kauf genommen. Diese werden sich im weiteren Verlauf dieses Programmierkurses aber alle auflösen.

Aufgabe 1

Geben Sie in Ihrem Editor den folgenden Quellcode ein, und speichern Sie die Datei im Ordner *JavaUebung01* unter dem Namen *Uebergabe.java*. Übersetzen Sie anschließend den Quellcode in den Bytecode, indem Sie den Compiler aufrufen. Beseitigen Sie eventuell gemeldete Fehler, bis der Übersetzungsvorgang erfolgreich ist, und testen Sie das Programm durch Übergabe an den Interpreter.

```
public class Uebergabe {
  public static void main (String[] args) {
    System.out.println("Der Parameter war: " + args[0]);
  }
}
```

Listing 1.7 Programm mit Übergabeparameter

Beim Starten des Programms kann ein Parameter übergeben werden.

Der Aufruf

```
java Uebergabe Hallo
```

erzeugt folgende Ausgabe:

```
Der Parameter war: Hallo
```

Soll der Parameter aus mehreren Wörtern bestehen, müssen Sie den ganzen Satz in Anführungszeichen setzen:

```
java Uebergabe "Mehrere Wörter müssen in Anführungszeichen gesetzt werden."
```

Den Parameter holt sich das Programm in den Platzhalter `args[0]`. Dadurch wird der Wert hinter dem Text `Der Parameter war:` ausgegeben. Das Programm erwartet auf jeden Fall einen Parameterwert. Wird beim Aufruf kein Parameter angegeben, wird eine soge-

nannte *Exception* (Ausnahme) erzeugt und eine entsprechende Fehlermeldung ausgegeben.

Aufgabe 2

Erstellen Sie im gleichen Projektordner *JavaUebung01* folgendes Programm mit dem Namen *Kreisberechnung*:

```
/* Kreisberechnung: Für einen Kreis mit dem Radius 5 cm
   werden der Umfang und die Fläche berechnet*/

public class Kreisberechnung {
  public static void main(String[] args ) {
    var radius = 5.0;
    var umfang = 2.0 * 3.1415926 * radius;
    var flaeche = 3.1415926 * radius * radius;
    System.out.print("Umfang: ");
    System.out.println(umfang);
    System.out.print("Fläche: ");
    System.out.println(flaeche);
  }
}
```

Listing 1.8 Das Programm »Kreisberechnung«

Das Programm berechnet für einen Kreis von 5,0 Längeneinheiten den Umfang sowie die Fläche des Kreises und zeigt anschließend die berechneten Ergebnisse in der Eingabeaufforderung an.

- Die Formel zur Berechnung des Kreisumfangs ist:

  $U = 2 \times \pi \times r$
- Die Formel zur Berechnung der Kreisfläche ist:

  $A = \pi \times r^2$

Dafür verwendet das Programm Zahlenwerte, Rechenoperatoren und Platzhalter (Variablen). Die Erläuterungen dazu folgen, wie weiter oben bereits erwähnt, in den nachfolgenden Kapiteln. Übernehmen Sie einfach den Quellcode so wie vorgegeben, und beachten Sie, dass Kommazahlen wie im englischsprachigen Raum mit dem Punkt als Dezimaltrennzeichen geschrieben werden müssen.

Die ersten zwei Zeilen werden durch die Zeichenfolge /* und */ eingeschlossen. Bei ihnen handelt es sich um Kommentare, die vom Compiler bei der Übersetzung ignoriert

werden. Mit Kommentaren kann der Programmierer seinen Quellcode für sich selbst als Gedächtnisstütze und für andere als Lesehilfe beschreiben und erläutern.

Aufgabe 3

Als dritte Übungsaufgabe erstellen Sie im Programmordner *JavaUebung01* das Programm *Kreisberechnung2*. In diesem soll die Übergabemöglichkeit von Parametern genutzt werden. Dem Programm soll als Parameter der Radius für den zu berechnenden Kreis übergeben werden, damit für einen beliebigen Kreisradius der Umfang und die Fläche berechnet werden können:

```
/* Kreisberechnung: Für einen Kreis werden der Umfang und der
 * Flächeninhalt berechnet.
 * Der Kreisradius wird beim Programmstart als Parameter
 * übergeben.
*/

public class Kreisberechnung2 {
  public static void main(String[] args) {
    var radius = Double.parseDouble(args[0]);
    var umfang = 2.0 * 3.1415926 * radius;
    var flaeche = 3.1415926 * radius * radius;
    System.out.print("Umfang: ");
    System.out.println(umfang);
    System.out.print("Fläche: ");
    System.out.println(flaeche);
  }
}
```

Listing 1.9 Das Programm »Kreisberechnung2«

Aufgabe 4

Das Programm von Aufgabe 3 soll so ergänzt werden, dass als zweiter Parameter die Einheit (z. B. `m` oder `cm`) übergeben und bei der Ergebnisausgabe verwendet wird. Speichern Sie das Programm ebenfalls im Programmordner *JavaUebung01* unter dem Namen *Kreisberechnung3*.

```
/* Kreisberechnung: Für einen Kreis
 * werden der Umfang und der Flächeninhalt berechnet.
 * Der Radius wird beim Programmstart als erster Parameter und
 * die Einheit wird als zweiter Parameter übergeben.
```

```
 */

public class Kreisberechnung3 {
  public static void main(String[] args) {
    var einheit = args[1];
    var radius = Double.parseDouble(args[0]);
    var umfang = 2.0 * 3.1415926 * radius;
    var flaeche = 3.1415926 * radius * radius;
    System.out.print("Umfang: ");
    System.out.print(umfang);
    System.out.println(" " + einheit);
    System.out.print("Fläche: ");
    System.out.print(flaeche);
    System.out.println(" " + einheit + '\u00b2');
  }
}
```

Listing 1.10 Das Programm »Kreisberechnung3«

Wie Sie dem Quellcode entnehmen können, wird ein zweiter Parameter mit dem Ausdruck `args[1]` angesprochen. Beim Programmstart werden die Parameter, durch eine Leerstelle getrennt, hinter dem Programmnamen angegeben. Die Ausgabe der Hochzahlen in der Einheit der Kreisfläche wird durch die etwas eigentümlich anmutende Angabe `'\u00b2'` erreicht. Auch diesem Mysterium werden wir später auf den Grund gehen.

Aufgabe 5

Das folgende Programm soll einen kleinen Einblick in die Möglichkeiten von Java liefern. Damit es korrekt funktioniert, benötigen Sie eine Bilddatei. Im Quellcode ist die Bilddatei mit dem Namen *java-logo.jpg* angesprochen. Verwenden Sie entweder diese Bilddatei aus dem Ordner *JavaUebung01* aus dem Downloadmaterial, oder wählen Sie eine beliebige andere Bilddatei, die dann im Programmfenster angezeigt werden soll. Kopieren Sie diese Bilddatei in den Ordner *JavaUebung01*, in dem sich auch Ihre Quellcodedatei befindet. Wenn Sie eine beliebige andere Bilddatei verwenden, sollten Sie daran denken, dass Sie im Quellcode den Dateinamen entsprechend anpassen müssen.

Erstellen Sie nun im Projektordner *JavaUebung01* das Programm *GrussMitProgrammfenster* mit dem folgenden Quellcode:

```
/* Beispiel mit Programmfenster
*/

import java.awt.*;
import javax.swing.*;

public class GrussMitProgrammfenster extends JFrame {
  public GrussMitProgrammfenster() {
    super("Hallo");

    var icon = new ImageIcon("java-logo.jpg");
    var label1 = new JLabel("Viel Erfolg beim", JLabel.CENTER);
    var label2 = new JLabel("Programmieren mit Java!", JLabel.CENTER);
    var label3 = new JLabel(icon);
    var schrift = new Font("SansSerif", Font.BOLD, 24);
    label1.setFont(schrift);
    label1.setForeground(Color.red);
    label2.setFont(schrift);
    label2.setForeground(Color.red);
    var c = getContentPane();
    c.setLayout(new FlowLayout());
    c.setBackground(Color.white);
    c.add(label1);
    c.add(label2);
    c.add(label3);
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    setSize(300,250);
    setVisible(true);
  }

  public static void main(String[] args) {
    new GrussMitProgrammfenster();
  }
}
```

Listing 1.11 Ein Programm mit Programmfenster

Die Ausgabe des jetzt schon etwas aufwendigeren Programms wird in einem Programmfenster erfolgen. Wenn Sie die Bilddatei *java-logo.jpg* verwendet haben, zeigt sich das Ergebnis wie in Abbildung 1.35.

Sie beenden das Programm wie gewohnt, indem Sie das Programmfenster über das Symbol X in der rechten oberen Ecke schließen.

Das Beispiel zeigt, dass mit Java auch sehr ansprechende Programme erstellt werden können, dazu aber wesentlich mehr Programmieraufwand erforderlich ist. Wir werden zum Erlernen der Programmiersprache Java zunächst mit Programmen beginnen, die zwar nicht ganz so ansprechende Ausgaben erzeugen, dafür aber einfacher zu erstellen sind und die erforderlichen Programmierkenntnisse überschaubar halten. Unser Ziel soll es allerdings sein, am Ende auch ansprechende Programme mit grafischer Oberfläche erstellen zu können.

Abbildung 1.35 Das Programmfenster zu Aufgabe 5

Aufgabe 6

Als letzte Übungsaufgabe dieses Kapitels erstellen Sie nochmals ein Programm zur Kreisberechnung. Radius und Einheit sollen in diesem Programm aber nicht als Parameter beim Programmaufruf übergeben werden, sondern die beiden Werte sollen über die Tastatur eingegeben werden, nachdem das Programm gestartet worden ist. Dazu soll das Programm jeweils einen Eingabedialog anzeigen, der zur Eingabe der betreffenden Angabe auffordert.

Die Eingabedialoge werden das Aussehen von Abbildung 1.36 und von Abbildung 1.37 haben.

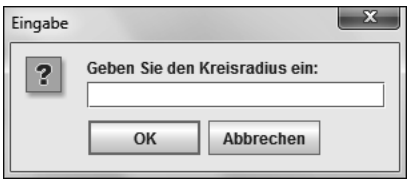


Abbildung 1.36 Der Eingabedialog für den Kreisradius

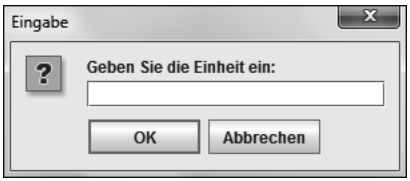


Abbildung 1.37 Der Eingabedialog für die Einheit

Erstellen Sie im Projektordner *JavaUebung01* das Programm *Kreisberechnung4* mit folgendem Quellcode:

```
/* Kreisberechnung: Der Radius für einen Kreis und die Einheit
 * werden über die Tastatur eingegeben. Anschließend
 * werden der Umfang und der Flächeninhalt berechnet.
*/

import javax.swing.*;

public class Kreisberechnung4 {
  public static void main(String[] args) {
    var eingabe = JOptionPane.showInputDialog("Geben Sie den Kreisradius ein: ");
    var radius = Double.parseDouble(eingabe);
    eingabe = JOptionPane.showInputDialog("Geben Sie die Einheit ein: ");
    var einheit = eingabe;
    var umfang = 2.0 * 3.1415926 * radius;
    var flaeche = 3.1415926 * radius * radius;
    System.out.print("Umfang: ");
    System.out.print(umfang);
    System.out.println(" " + einheit);
    System.out.print("Fläche: ");
    System.out.print(flaeche);
    System.out.println(" " + einheit + '\u00b2');
  }
}
```

Listing 1.12 Das Programm »Kreisberechnung4«

1.5 Ausblick

Das erste Kapitel hat Ihnen einige Informationen zu Java als Programmiersprache geliefert. Sie haben einiges über die unterschiedlichen Arbeitsweisen von Programmier-

sprachen erfahren und können Java jetzt in diesem Gesamtkontext einordnen. Außerdem haben Sie Hinweise zur Installation von Java in Ihrer Arbeitsumgebung erhalten, und Sie haben etwas über die verschiedenen Betriebsmodi Ihrer Java-Umgebung gelernt. Damit sind die Voraussetzungen für einen Start in die praktische Arbeit geschaffen. Sie kennen nun den Aufbau einfacher Java-Programme und wissen, wie Sie bereits mit minimalen Hilfsmitteln Java-Programme erstellen und auf Ihrem Computersystem starten können.

In diesem Kapitel habe ich noch auf die Unterstützung durch ein Entwicklungssystem verzichtet, damit Sie auch einen Einblick in die Abläufe erhalten, die sich bei der Verwendung einer Entwicklungsumgebung im Hintergrund verborgen abspielen. Ein Verständnis dieser Abläufe ist immer dann hilfreich, wenn etwas nicht nach Ihren eigenen Vorstellungen funktioniert und Sie Abhilfe schaffen wollen. Zudem werden Sie die Vorteile einer Entwicklungsumgebung auch besser zu schätzen wissen, wenn Sie die Erfahrung gemacht haben, welche Arbeitsabläufe durch diese vereinfacht werden.

Im folgenden Kapitel werde ich Sie mit den Java-Grundlagen vertraut machen. Sie werden wichtige Bausteine eines Java-Programms kennenlernen. Dazu gehören Bezeichner und Schlüsselwörter, die in Java eine spezielle Bedeutung haben. Zudem zeige ich Ihnen unterschiedliche Möglichkeiten, um Ihren Programmcode zu kommentieren, und Sie erfahren, wie Sie einfache Datentypen und Operatoren verwenden können, um die Rechenleistung des Computers zu nutzen.

Inhalt

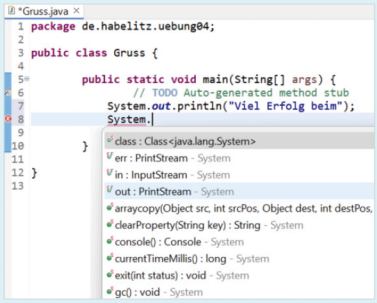
Programmieren lernen mit Java

»Ideal für Selbststudium und Ausbildung«

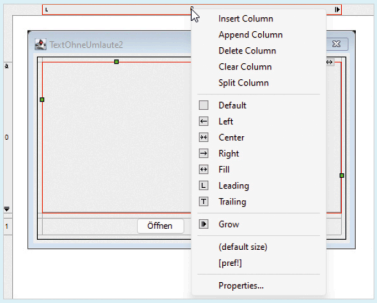
Schritt für Schritt zum ersten Java-Programm

Sie möchten richtig Programmieren lernen? Steigen Sie einfach ein! Trauen Sie sich an Objekte, Algorithmen und Datenbanken heran. Mit diesem Buch meistern Sie auch die anspruchsvolleren Programmiertechniken. Und schon bald wird aus Ihrem ersten Dreizeiler eine komplette Ampelsteuerung mit Threads.

Installieren und loslegen



Viele Beispiele und Übungen



Eigene Programme entwickeln

So gelingt der Einstieg

Nach einer ausführlichen Anleitung richten Sie Ihre Arbeitsumgebung ein und erstellen Ihr erstes Programm. Schritt für Schritt erlernen Sie die Grundlagen der Programmierung und steigen in Java ein.

Entwickeln Sie komplette Java-Anwendungen

Sprechen Sie mit dem Computer – aber auch mit Ihren Anwendern! Erstellen Sie Dialoge, Fenster und Schaltflächen. Geben Sie Ihren Nutzerinnen und Nutzern Feedback und stellen Sie Daten übersichtlich dar.

Übung macht den Meister

In jedem Kapitel gibt es viele Übungen, mit denen Sie das Programmieren trainieren. Auch die Codebeispiele laden Sie ein: Probieren und variieren Sie, spielen Sie mit den Effekten!

Alle Codebeispiele sowie Musterlösungen für die Übungsaufgaben stehen zum Download bereit.

Dipl.-Ing. Hans-Peter Habelitz unterrichtete Informatik an einer berufsbildenden Schule. Er hat schon vielen Anfängern das Programmieren beigebracht und als Dozent für Fachdidaktik der Informatik sein Know-how weitergegeben.

Auf einen Blick

Grundlagen
Java und Eclipse
Von der Idee zum Code
Bedingungen und Schleifen
Java-Sprachelemente

Mit Objekten arbeiten
Klassen, Objekte, Methoden
Zugriffsschutz richtig setzen
Fehler und Ausnahmen
Multithreading einsetzen
Datenbanken mit JDBC
Auf Dateien zugreifen

Grafische Benutzeroberflächen
Fenster, Schaltflächen & Co.
GUIs mit dem WindowBuilder
Animationen erstellen
Datensätze präsentieren

Rheinwerk Computing

Gedruckt in Deutschland
Mineralölfreie Druckfarben
Zertifiziertes Papier

€ 24,90 [D] € 25,60 [A]
Für Windows, macOS und Linux
Programmierung
ISBN 978-3-367-10478-9